XDD

User's Guid

Version 7.1
May 2010

I/O Performance, Inc

Principle Author:	Thomas M. Ruwart *tmruwart@ioperformance.com*
Contributing Authors:	Steve Hodson, DoE/ORNL *hodsonsw@ornl.gov* Steve Poole, DoE/ORNL *spoole@ornl.gov* Bradley Settlemyer, DoE/ORNL *settlemyerbw@ornl.gov* Russell Cattelan, Digital Elves *cattelan@thebarn.com* Alex Elder
Phone:	612-850-2918
Email:	tmruwart@ioperformance.com

Change History:

The revision numbers have the following meaning. The "number" such as 5.7, 5.8, 5.9, …etc. represent major release changes such as the addition of new options or sections to XDD.

Revision	Date	Author	Description of Changes
7.0	18 JAN 2010	Thomas M. Ruwart	Release of 7.0
7.0 Draft3	24MAR2010	Thomas M Ruwart	Updates – preallocate, cleanup
7.1 Draft1	24May2010	Thomas M Ruwart	Updates to be consistent with version 7.1

Table of Contents

1 Introduction

1.1 About this document

This document is a user's guide to compiling, installing, and running the XDD program. It also has a variety c
examples and hints for understanding the use and results of XDD measurements.

1.2 Acknowledgements

The continuing work on XDD is supported by funding and resources of the National Center for Computationa
Sciences (NCCS) at Oak Ridge National Laboratory (ORNL), the Extreme Scale Systems Center (ESSC) at
ORNL, and the Department of Defense.

1.3 About XDD

XDD is a tool for measuring performance and characterizing disk subsystem I/O behavior on single systems a
clusters of systems. It is a command-line tool that grew out of the UNIX world and has been ported to run in
Window's environments as well. It is designed to provide consistent and reproducible performance measurem
of disk I/O traffic. XDD is a single, self-contained program that can be run on a single system with no other
dependencies. There are two other tools included with the XDD package called timerserverr and gettime that c
be used in conjunction with XDD that are used to synchronize the clocks of XDD programs simultaneously
running across multiple computer systems. However, timeserver and gettime are not required to run XDD.

As of XDD Release 7.0 it is possible to use XDD to copy files from one computer to another over a network.
supporting shell script is also provided in the "contrib" subdirectory to assist in using this capability. See the E
to-End option and examples for more information on this feature.

1.4 License Agreement

1.5 Distribution of XDD

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Pu License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

1.6 Versioning

XDD is distributed as a base version (e.g. 7.0.MMDDYY) and a particular "build" number (hhmm). Subseque builds using "make" will produce new version dates (MMDDYY) and build numbers. The XDD "-version" option will display the base version number and the version being run.

Example: **`XDD.Linux.7.0.011510.Build.1504.tgz`**

1.7 System Requirements

XDD can run on a relatively minimally configured system but it is recommended that adequate main memory capacity and processor speed be available when using some of the more advanced features of XDD. The XDD program itself is not CPU intensive but the faster the processor and the less loaded a system is the more accura and consistent the results will be. As a rule, a minimum of 128MB of main memory is recommended. All othe system parameters are left to the discretion of the user.

2 What's new

2.1 In 7.0

This release of XDD is based on what was supposed to be release 6.7 that was never officially released. There are several major changes in release 7.0 that include:

- Major restructuring of the source code for readability, extensibility, and supportability
- Addition of an end-to-end option that allows XDD to quickly and reliably copy data from one comput system to another while measuring the end-to-end performance of the operation
- A "restart" monitor used in conjunction with the end-to-end operations to resume a failed copy operat
- A "results" manager thread that is used to collect and display results from all I/O threads
- The ability to selectively reformat the output fields (-outputformat option)

2.2 In XDD 6.7

This section is included for reference purposes only since this version was never actually released. There were some major source code and program structure enhancements in 6.7 that were carried forward and improved u in release 7.0. This was also when the new build procedure and associated "Makefile" was introduced. The following is a summary of what happened in 6.7:

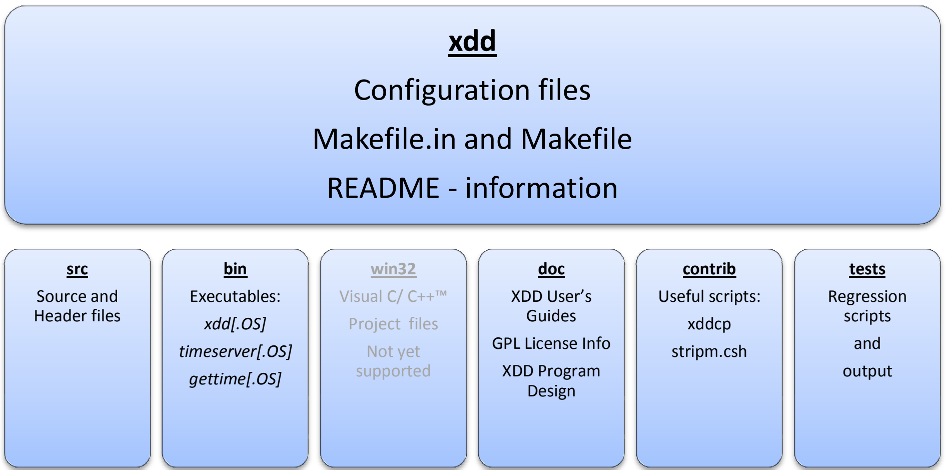
- The "-align" option has been renamed to "-memalign" to more accurately reflect the meaning of this option.
- The "-delay" option has been renamed to "-passdelay" to more accurately reflect the meaning of this option.
- SCSI Generic device recognition has been added so that the "-sgio" option is not required when specifying an SG device as a target (i.e. –target /dev/sg2). The "-sgio" option is required if for some reason the name of the SG device is not "sg#" where # is the device number.
- The –preallocate option works on Linux XFS

3 Compiling and Installing XDD

3.1 XDD Source Code Overview

The XDD distribution comes with all the source code necessary to install XDD and the companion programs timeserver and gettime. There are also a variety of scripts that can be used for testing and moving data. The X User Guide is also included as a PDF. The "win32" subdirectory contains the Visual C/C++ project files but a this release the Windows version is not yet supported.

The XDD distribution directory hierarchy has the following structure:



3.2 Detailed Description of Source and Header files

There are three programs that make up the standard XDD distribution:

1. XDD – xdd.c
2. timeserver – timeserver.c
3. gettime – gettime.c

The XDD program uses most of the subroutines contained in the distribution but timeserver and gettime share some of these subroutines. Many of the subroutines have corresponding header files as noted in the following descriptions.

3.2.1 xdd.c

xdd.c contains all the program routines that are specific to running XDD.
.

3.2.2 timeserver.c

timeserver is the master time server that runs on a single machine and is used as a reference clock by XDD programs running on other machines. This program does not have to be compiled in order to compile/run XDI a single machine.

3.2.3 gettime.c

gettime is a program that is used in conjunction with the time server functions when running XDD on multipl machines (see section on Timeserver Functions on Multiple Machines). This program does not have to be compiled in order to compile/run XDD on a single machine.

3.2.4 access_pattern.c and access_pattern.h

These subroutines generate the seek list used by XDD to access a target.

3.2.5 barrier.c and barrier.h

This contains all the barrier subroutines that are used to control the execution and timing of I/O operations.

3.2.1 datapatterns.c and datapatterns.h

This contains all the barrier subroutines that are used to control the execution and timing of I/O operations.

3.2.2 end_to_end.c and end_to_end.h

These files contain most of the routines used to implement the "-endtoend" operations. Some other end_to_en routines can be found in initialization.c.

3.2.1 global_clock.c

Contains all the functions that initialize the global clock network (see also global_time.c).

3.2.1 global_data.c and global_data.h

Contains all the functions necessary to initialize the global data structure.

3.2.2 global_time.c

Contains all the functions necessary to contact the master time server machine and establish the Global Time.

3.2.3 heartbeat.c

Contains all the code used to implement the heartbeat thread.

3.2.1 info_display.c

Contains the functions necessary that display system, device, and software configuration for a run.

3.2.2 initialization.c

This file contains most of the subroutines used during program and target thread initialization.

3.2.1 interactive.c, interactive_func.c, interactive_table.c, and interactive.h

These files contain the subroutines used in the main loop of a run. A graphical description of these routines is provided in the section on Overall Program Structure.

3.2.1 io_buffers.c

Contains the functions necessary to that allocate and initialize the memory buffers used for IO.

3.2.1 lockstep.c and lockstep.h

The lockstep.c function contains the various subroutines specific to the "-lockstep" option. The lockstep.h file contains the data structure that is used by the subroutines in lockstep.c. The lockstep_t data structure is pointe by a member in the PTDS.

3.2.1 memory.c

This file contains the memory locking and unlocking functions.

3.2.1 misc.h

misc.h contains all the miscellaneous definitions that are common to all the programs.

3.2.1 nt_unix_compat.c

nt_unix_compat.c contains all the UNIX subroutines that are not supported in a standard Windows™ NT™ or Windows™ 2000™ environment. These subroutines provide a mapping from the UNIX system call (i.e. sleep getpid, pthread_create, …etc.) to the equivalent Windows™ system call (i.e. Sleep, GetCurrentThreadID, CreateThread, …etc.). This file is only used when compiling in a Windows™ environment. It is not part of a UNIX compile.

3.2.2 parse.c, parse_func.c, parse_table.c, and parse.h

This file contains the subroutines that parse the command-line options and the setup file if it is specified. Pars contains the main parsing logic. Parse_table.c contains the table of valid options and references to the functio that perform option-specific parsing. These functions are all located in the parse_func.c file. Each option-spec parsing function has a name that follows the following naming convention:

```
xddfunc_<option name>()
```

Where <option_name> is simply the name of the option. For example, for the “-verbose” option, the associate option-specific function is called “**`XDDfunc_verbose`**()”. And so on for all the other functions.

3.2.3 pclk.c and pclk.h

pclk.c contains all the subroutines provide pico-second clock values to the calling function.

3.2.1 preallocate.c

This file contains the subroutines required to implement the “-preallocate” option.

3.2.1 processor.c

This fine contains the subroutines necessary to implement the “-processor” option.

3.2.1 ptds.c and ptds.h

This file contains the subroutine that initializes a PTDS structure as defined in ptds.h.

3.2.1 qthread.c and qthread_*.c

The qthread.c file contains the main entry point for the “QThread” pthread that runs under the direction of a Target Thread pthread. The other qthread_*.c source files perform initialization, cleanup, generic and OS-spec I/O calls, and things to do (i.e. ttd) before and after I/O operations.

3.2.2 read_after_write.c and read_after_write.h

This file contains the subroutines that control a read-after-write operation.

3.2.3 results_display.c, results_manager.c, and results.h

The file results_display.c contains all the subroutines that implement the defined display format identifiers. Th file results_manager.c contains all the subroutines that make up the results_manager thread that is used to pro and display the results information.
results.h contains the definition of the “results” structure and associated flag definitions.

3.2.1 schedule.c

This file contains the subroutines that implement various scheduling options.

3.2.2 sg.c and sg.h

These files contain all the subroutines required to implement SCSI Generic I/O under Linux.

3.2.1 target.c

This file contains the subroutines that perform initialization functions such as starting the QThreads and openi
the target device/files.

3.2.2 ticker.c and ticker.h

ticker.c contains all the subroutines that are specific to a specific machine architecture that are required to acc
that's machine's high resolution clock.

3.2.1 signals.c

This file contains the signal handler for XDD.

3.2.1 Target_thread.c and target_*.c

The target_thread.c file is the main entry point for the Target Thread pthread. The other target_*.c source files
perform initialization, cleanup, all I/O operations for a single pass, and things to do (i.e. ttd) before and after a
"pass" and before an I/O operation is assigned to a QThread.

3.2.2 time_stamp.c and time_stamp.h

This contains the time stamp subroutines.

3.2.1 utils.c

This file contains a variety of utility functions used by other subroutines within XDD but are not specific to ar
one option or subroutine.

3.2.2 verify.c

This contains all subroutines required to implement the "–verify" option.

3.2.3 xdd* header files

The xdd.h header file is common to xdd.c and to all the other programs that use XDD subroutine files. The xd
"includes" all the other xdd_* header files in the proper order:
xdd_base_version.h
xdd_[OS].h such as xdd_linux.h, xdd_aix.h, …etc for all supported Operating Systems
xdd_common.h – OS-independent
xdd_prototypes – the prototype definitions of all XDD subroutines
global_data.h – definition of the XDD Global Data Structure

The xdd_base_version.h file contains the version number and is created by the Makefile each time a new
distribution of XDD is built. To make a new version the "XDDVERSION" variable in the Makefile is change
and the xdd_base_version.h header file is built by issuing a "make baseversion" command.

3.3 Building XDD

As of release 7.0 the XDD distribution comes with an “auto-configuration” script. The overall process of buil
XDD and related programs consists of running “configure” followed by “make” which will create the OS-appropriate executables and place them in the local “bin” subdirectory. It is important to note that the standar
XDD distribution no longer contains any pre-compiled executables.

3.3.1 Configure

After has been run and the files are “clean”, it is safe to make XDD, the timeserver, and the gettime programs.
The process has been simplified for most XDD-supported operating systems.
From the directory that contains the script called “configure” run the following command:

```
./configure
```

This will build the file called “Makefile” that contains all the OS-dependencies adjusted for the build.
At that point run any of the following:

```
make
```

or

```
make install
```

or

```
make clean ; make
```

A simple “`make`” will simply make the xdd, timeserver, and gettime executables in the local XDD bin subdirectory.
The “`make install`” will attempt to copy the XDD, timeserver, and gettime executables into /`sbin` on th
local system. The user must have root privileges in order to perform a “`make install`”.
The “`make clean`” is used to remove all object files and previously compiled executables from the local X
bin directory. It must be followed by an explicit “`make`” or “`make install`” to recompile the executables.

3.4 Supported Operating Systems

XDD is currently supported on a number of mainstream operating systems with limited support for some lega
operating systems. These operating systems include:

- Linux (kernel versions 2.6 and above only)
- AIX™ from IBM
- Mac OS X
- FreeBSD
- Solaris™ from Sun on Intel platforms
- Windows™ 2000™, Windows™ XP™, Windows™ Vista™, Windows™ Server™ 2003™, Window
  Server™ 2008™, Windows™ 7™
- Limited support for legacy operating systems:
  - IRIX™ from SGI
  - Solaris on SPARC platforms

The process for building XDD is relatively straight forward for all operating systems. There are two basic buil
environments: Windows™ systems and Unix-like systems. The basic process of building XDD is to extract th
files from the XDD distribution archive and run the build program. The build program for Windows is Visual C++™. The build program for Unix-like systems is “make” plus the “c” or “gcc” compiler. In either case, the XDD, timeserver, and gettime executables are built and placed in the “bin” subdirectory (see diagram).

For all Unix-like systems that there will be an executable with and without the operating system name as an extension. For example, on a Linux system, the build process will produce an executable file called “xdd.Linu
and one called “XDD”. The “XDD” executable is simply a hard link to the “xdd.Linux” executable. The same applies to the “timeserver” and “gettime” executable files. In other words, at the successful conclusion of the build process, the “bin” directory will contain the following files:

- “xdd.<OS>” and a hard link to “xdd”
- “timeserver.<OS>” and a hard link to “timeserver”

- “gettime.<OS>” and a hard link to “gettime”

  Where <OS> is either Linux, OSX, FreeBSD, or Solaris respectively.

Finally, it is important to note that XDD does not depend on the “timeserver” or “gettime” programs. These additional programs are used in multi-hosted environments.

3.5 Windows™

The distribution directory hierarchy includes a subdirectory called “win32”. This subdirectory contains the Vi C/C++™ Project File (called XDD.vcproj) that can be used to recompile XDD for most any flavor of Windows™. The source files and header files are assumed to be one level above the “win32” subdirectory. Th is a “debug” directory in the “win32” subdirectory that contains all the intermediate files that Visual C/C++™ uses during a build operation. The executable XDD file (*xdd.exe*) is created in the “bin” directory just above t “win32” subdirectory. The *xdd.exe* program is sufficient to run on most any Windows™ platform. Use of any other compiler can produce unpredictable results.

At the successful conclusion of the build process, the “bin” directory will contain the following files:

- “xdd.exe”
- “timeserver.exe”
- “gettime.exe”

Warning: The presence of “.exe” files in any kind of an archive may prevent the archive from being sent thro most email systems.

3.6 Build Notes

Before building XDD on any of the UNIX-based systems, it *may* be necessary to “clean” the files first. Since t XDD distribution is sometimes built on a Windows platform, certain Windows artifacts may contaminate the source, header, and make file. More specifically, any text file (such as *Makefile*, *.c* and *.h* files) may contain he characters “0D” (ctl-M) at the end of each line. This is not interpreted correctly in most UNIX environments makes it impossible to compile any of the XDD source code. Therefore, these hex “0D” characters must first b removed before any compilation can be done.

A simple script in the XDD local “contrib” subdirectory is provided to perform this removal operation automatically. The script is *stripm.csh*. Simply run the *stripm.csh* script and it will fix all affected files. At the UNIX command prompt, use the following command to execute *stripm.csh*:

```
root#   csh stripm.csh
```

Alternatively, the following commands can be given to the C-shell or shell:

C-Shell example:

```
foreach x ( *.c *.h *.makefile )
 tr -d '\r' < ${x} > y
 mv -f y ${x}
 echo " " >> ${x}
end
```

Shell example:

```
for x in *.c *.h *.makefile
do
 tr -d '\r' < ${x} > y
 mv -f y ${x}
 echo " " >> ${x}
done
```

3.7 Notes about Solaris™ and Open Solaris from Sun

There is an issue with running XDD on Solaris that involves the default number of semaphores that Solaris all
each process to use. It is normally too low for XDD to run and XDD generates errors that say things like “can
allocate barrier for …, not enough space”. The following parameters can be put in /etc/system on a Solaris sys
and a reboot will take care of this problem:

```
     set semsys:seminfo_semmni=4096
     set semsys:seminfo_semmns=8192
     set semsys:seminfo_semmap=4098
     set shmsys:shminfo_shmmni=512
     set shmsys:shminfo_shmseg=32
```

These settings may be a little high and can be adjusted to meet the local system constraints.

4 Theory of Operations

4.1 XDD Program Structure Overview

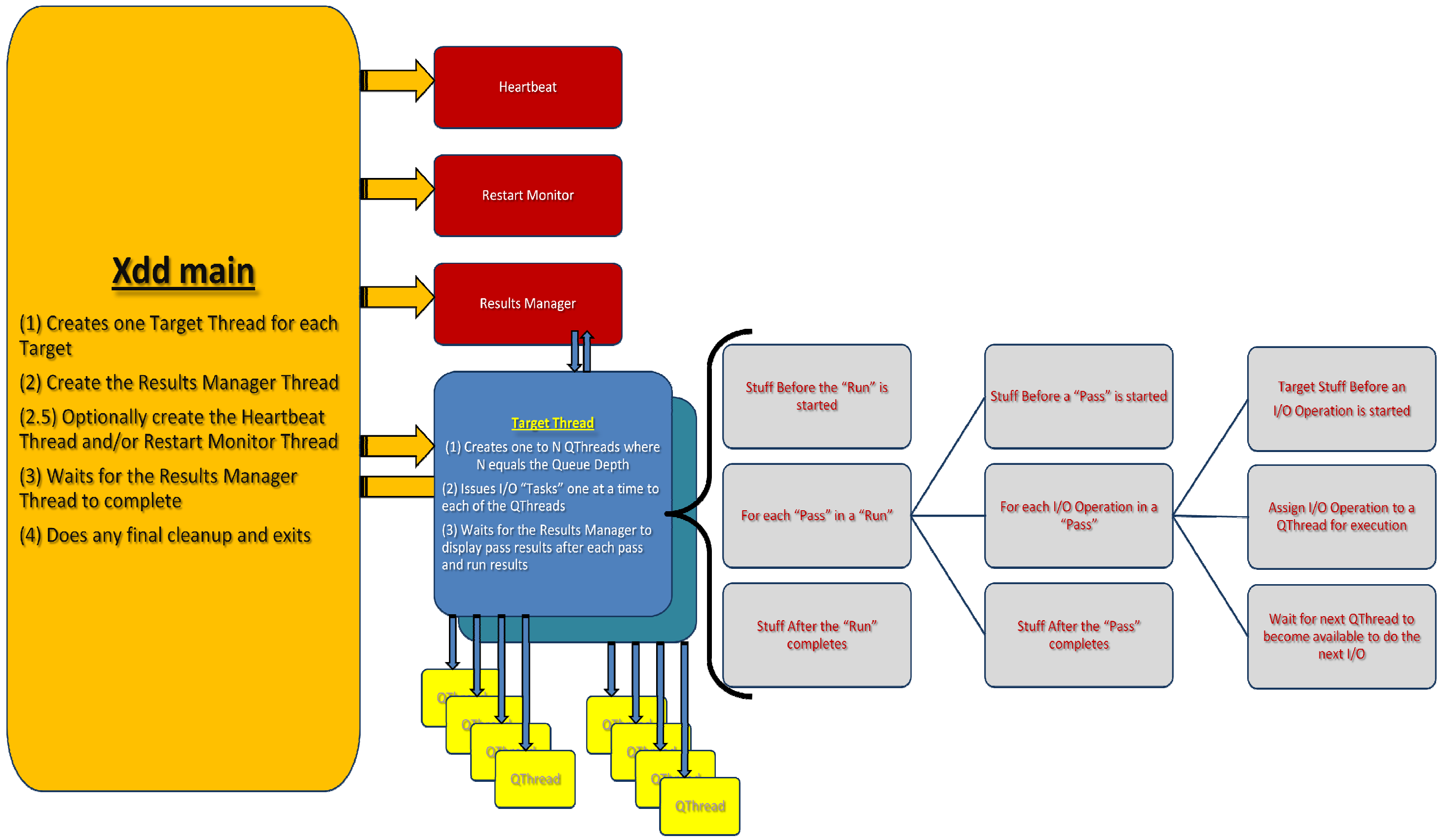
XDD consists of the main XDD executable and one or more “target” threads that are created to perform the ac I/O operations on a specific target device or file. The following terms are important to understand in order to make sense of the program and source code structure:

- The time from the invocation of the XDD command until it returns control to the shell is referred to as “run”
- Each “run” consists of one or more “passes”
- Each “pass” consists of one or more I/O “operations”

For example, the following simple XDD command line will perform an XDD “run” that consists 5 passes of 1 write I/O Operations (4Kbytes each) to be performed on the specified “target” called “testfile”. The “verbose” option will cause XDD to display intermediate results for each “pass” as the pass completes.

```
XDD –op write –target testfile  -reqsize 4 –numreqs 128 –passes 5 –verbose
```

The overall program structure has been simplified the several basic parts as shown in Figure 1.



Xdd main
(1) Creates one Target Thread for each Target
(2) Create the Results Manager Thread
(2.5) Optionally create the Heartbeat Thread and/or Restart Monitor Thread
(3) Waits for the Results Manager Thread to complete
(4) Does any final cleanup and exits
Heartbeat
Restart Monitor
Results Manager
Target Thread
(1) Creates one to N QThreads where N equals the Queue Depth
(2) Issues I/O "Tasks" one at a time to each of the QThreads
(3) Waits for the Results Manager to display pass results after each pass and run results
QThread
QThread
Stuff Before the "Run" is started
For each "Pass" in a "Run"
Stuff After the "Run" completes
Stuff Before a "Pass" is started
For each I/O Operation in a "Pass"
Stuff After the "Pass" completes
Target Stuff Before an I/O Operation is started
Assign I/O Operation to a QThread for execution
Wait for next QThread to become available to do the next I/O

Figure 1 XDD Program Structure

4.2 Basic Operation

XDD is a program that performs data transfer operations between memory and a storage device or fil (or multiple storage devices / files) and collects and displays performance information about these I/ operations. Each device or file under test is referred to as a “target”. XDD creates one thread for each target referred to as Target Threads. Each Target Thread creates one or more QThreads to perform I/ the actual target device/file. I/O operations are either a read, write, or no-op operation of a fixed size known as the "request size". An XDD “run” consists of several “passes”, the number of which is specified by the “–passes” option. Each pass will execute some number of I/O requests on the specifi target(s) at the given request size. In general, each pass is identical to the previous passes in a run wit respect to the request size, number of requests to issue, and the access pattern unless certain options a used to alter these parameters between passes. Passes are run one after another with no delays betwe passes unless a pass delay is specified with the “–delay” option.

Multiple passes within an XDD run are used to determine the reproducibility of the results. In theory the results from each pass in an XDD run should be the same or at least very close to the same given same set of run-time parameters.

Upon invocation, XDD will parse the command line arguments and spawn one Target Thread for eac target specified for the run. The Target Thread will spawn a minimum of one QThread and up to the number of QThreads specified by the “–queuedepth“option. After the Target Threads have been initialized, they will wait at an “initialization barrier” until XDD “main” has spawned and initialized Results Manager thread and optionally the Heartbeat and/or the Restart Monitor threads which depen on the –heartbeat and –restart options respectively.

Once all the Target Threads and support threads have initialized, the first pass will start to run. All Target Threads start at the same time. At the conclusion of each pass, the Target Threads wait for the Results Manager to collect, process, and display the intermediate results of the prior pass and then st the subsequent pass, again, all at the same time.

At the conclusion of the final pass in a run, all the Target Threads will wait for the Results Manager t display the “pass” and “run” results. At this point the Results Manager will display the “average” res for each target as well as a “combined” average across all targets. After the Results Manager has displayed all the results it will release the Target Threads to perform any termination processing. On the Target Threads (and associated QThreads) have terminated, the Results Manager will enter a barr where the XDD “main” program thread is waiting. Once the Results Manager and XDD “main” are p this “final” barrier, they both exit.

4.3 Command Line Options and the Setup File

XDD has a command-line interface that requires all the run-time parameters to be specified either on XDD invocation command line or in a “setup” file. The format of the setup file is similar to the XDD command-line in that the options can be simply entered into the setup file the same as they would be seen on the command line. The following example shows an XDD invocation using just the comman line and the same invocation using the setup file along with the contents of the setup file.

Command line:
```
     xdd –op read –targets 1 /dev/scsi/disk1 –reqsize 8 –numreqs 128
     –verbose
```
Using a Setup file:
```
     xdd –setup XDDrun.txt
```

Where the setup file `XDDrun.txt` is an ASCII text file that contains the following:

```
–op read –targets 1 /dev/scsi/disk1
–reqsize 8
–numreqs 128
–verbose
```

4.4 Operation Specification and Request Sizes

The operation to perform is specified by the "–**op**" option. This can be either "*read*" or "*write*". This version of XDD *will* mix read and write operations within an XDD run according to the read/write ra set using the **–rwratio** option. Each r/w operation will transfer a given amount of data known as the "request size". The request size is specified by the "–**reqsize**" option in units of "blocks". A block is default, 1024 bytes but can be specified to be any positive integer value by using the "–**blocksize**" option. The "block" is used as the basic unit for all other options requiring a data size unless otherwis noted. It is recommended that the block size be specified as numbers that are integer multiples of 512 bytes (i.e. 1024, 2048, 5120, …etc.) since this tends to be the predominant sector size for most storag devices at the current time.

4.5 Target Specification and Multiple Target Synchronization

All requests are sent to a "target" which can be either a disk device or a file. A single XDD run can operate on a single target or multiple targets simultaneously. Target names are specified using the "–**targets**" option. It is always necessary to specify the number of targets followed by the individual tar names. In order to simplify the list of target names, the "**-targetdir**" option can be used to specify the directory where the target devices or files reside.

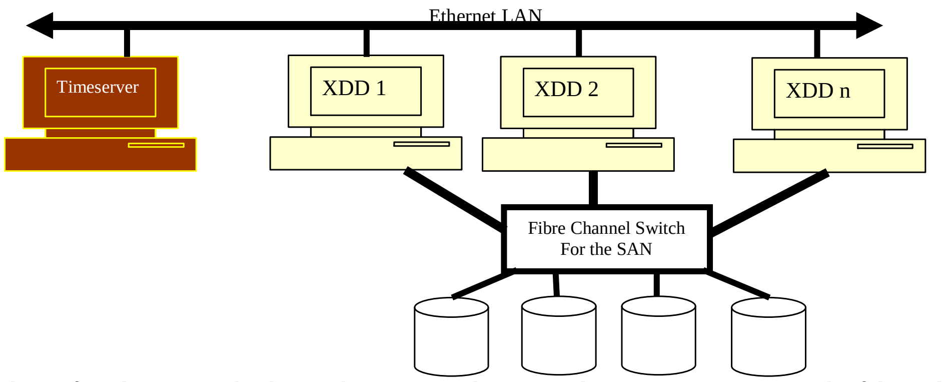
The execution of the XDD threads on multiple targets is synchronized through the use of "barriers". Each XDD I/O thread initializes itself and waits for all the other XDD I/O threads to "reach at startin point". Once all the XDD threads reach this point, they are all released simultaneously. Each XDD thread will run independently until it has either completed all of its requested I/O operations or reach its time limit (as specified by the –**timelimit** input parameter). At this point the XDD threads resynchronize with one another and begin another pass.

It is also possible to resynchronize the threads at specific points within a pass by using the "–**syncio**" and/or "**-syncwrite**" options. The **-syncio** option instructs each of the I/O threads to synchronize afte some number of I/O operations specified as an argument to this option. The default is to sychnronize only at the beginning of each pass. The –**syncwrite** option is used to synchronize "buffered" write operations at the end of each pass by flushing all the file system data buffers to the physical media. T option is not able to flush and cache buffers on the disk controllers or disk drives themselves.

Synchronizing the XDD I/O threads is done to insure that all the XDD I/O threads start at precisely t same time in order to avoid misleading results due to skewing the start times. It is possible to elimina synchronization by specifying the -**nobarrier** parameter. The result is that many of the I/O start time can be significantly skewed from one another for all participating XDD I/O threads. However, that m in fact be the desired effect.

4.6 Time Synchronization Across Multiple Computer Systems

Synchronization can also be done across multiple machines each running XDD. This is accomplished using the timeserver and gettime programs. The timeserver program provides a single global referenc clock that is used by the time stamping in order to be able to more accurately correlate events in time from multiple computer systems. The timeserver program should be run on a single machine as a background task. This machine must be accessible via a LAN to all the machines that will be running XDD. This LAN should preferably be a lightly used LAN for optimum results.



The preferred way to make this work is to use “ssh” to start the XDD programs on each of the nodes using the “-starttime” and “-timeserver” options. The “-timeserver” option tells the XDD program th host name or IP address of the time server machine. The XDD program will contact this machine to determine the “global” time that all the other XDD programs will use as a frame of reference. The “-starttime” option specifies the time to start in “global time” units.

Before running the “ssh XDD” command line in a script, it is necessary to determine a global time sometime in the future at which all the XDD program will start. The way to do this is to use the “gettime” program to contact the time server, determine the global time, add a specified number of seconds, and display the result on standard out. This global start time can then be used as the argume to the “-starttime” option for each of the “ssh XDD” commands. An example script would look like

```
….
set g=`gettime –timeserver 192.10.11.12 –add 20`
# At this point ${g} will be the current global time plus
# 20 seconds.
foreach i ( 1 2 3 4 )
        ssh host${i} XDD –starttime ${g} –op read –targets 1 /dev/dsk/c1d2s0 –mbytes 5
        timeserver host 192.10.11.12 …&
end
wait
…
```

This example scriptlet will set the local variable “${g}” to the global time plus 20 seconds. This valu then passed to each of the XDD programs that are started on host1, host2, host3, and host4. This will result in each XDD program starting at exactly the same time. However, if any or all of the host machines running XDD are in the presence of a black hole, neutron star, or other extremely massive body, the relativistic effects on the space-time continuum may produce unpredictable results.

4.7 XDD Run Time or Run Length

It is necessary to specify a limit on how *long* XDD will run. There are several ways to do this. First, i possible to explicitly specify the number of transfers to perform using the “**–numreqs**” option. This specifies the number of read or write calls to make for a single “pass”. It is also possible to simply specify the number of MegaBytes to transfer using the “**–mbytes**” option. It is important to note that MegaByte in this context is 1048576 bytes. Finally, it is possible to specify a time limit for each XD pass using the **–timelimit** option. This will cause each pass to terminate after the specified number of seconds has elapsed or after executing the specified number of requests or transferring the specified number of megabytes whichever occurs first.

4.8 I/O Range

For random I/O operations, each XDD I/O thread performs its I/O operations within a certain consecutive "range" of blocks on the target (see Figure 1). The range is either *implicitly* specified as number of MegaBytes to transfer or *explicitly* specified using the "–**range**" option. For example, if th user specifies 2048 purely sequential data transfers at a request size of 128 blocks each, then the rang will be implied as 262144 blocks (2048 * 128). However, if the user wants to transfer the same 2048 requests randomly over a 2 GigaByte area (or range) on the target then the range needs to be explicit specified as 268435456 blocks or 2 GigaBytes.

The beginning of the I/O range defaults to the beginning of the target whether it is a device or a file. possible to specify a "starting offset" such that the I/O range begins at some distance into the target (Figure 1). This can be done several ways. First, the "–**startoffset**" option can be used to start I/O operations at some distance into the target for an XDD run. The "–**passoffset**" can be used to incrementally move the starting offset further into the target device on subsequent passes within an XDD run by some specified number of blocks. Finally, the "–**targetoffset**" can be used to move the starting offset into the target device by a number of blocks that is determined by the target offset valu times the target's ordinal number. For example, given an XDD run on 4 targets with a target offset o 1024 and a starting offset of 0, I/O will start at block 0 on target 0, block 1024 on target 1, block 204 on target 2, and block 3072 on target 3.

4.9 Access Patterns

The range can be specified to start anywhere within the target so long as care is taken to insure that th end of the range is still within the confines of the target. This is particularly true when randomly accessing blocks within a target that is a regular file. Within the I/O range, data access patterns can b either

1) purely sequential
2) staggered sequential
3) purely random

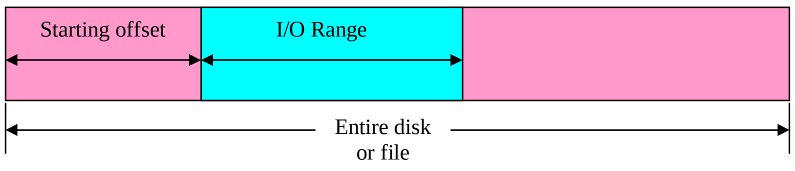


Figure 1. Example of the I/O range and the offset.

Purely sequential I/O is the default access pattern. This access pattern accesses consecutive data bloc starting at the beginning of the range to the end of the range.

A Staggered Sequential access pattern also starts at the beginning of the range and ends at the end of range but only transfers every Nth data block. This access pattern is specified by the "-seek staggered parameter. For example, if reading 256 MegaBytes within a 2 GigaByte range, the 256 MegaBytes is spread evenly over the entire 2GB range. Therefore, for a given request size, every 8th data block is r with gaps of unread data in between since 256 MegaBytes is 1/8th of 2 GigaBytes.

A purely Random access pattern is one that accesses data blocks at the specified request size random throughout the specified range. It is important to note that for a given range and given request size, th same random pattern is generated for each successive pass/run in order to yield *reproducible* result The random access pattern used for each pass within a run can be changed by using the "-randomize" parameter. The random access pattern used for each run can be changed by using the "-seek seed" parameter.

It should be noted that when running *XDD* on a regular file especially in "random" mode, the Direct option (-dio) should be used to avoid using the file system buffer cache. The file system data caching mechanisms will produce misleading (and non-reproducible) results. See the section on Reading and Writing Devices and Files for more information.

Finally, it is possible to specify a "null" access pattern whereby the first block in the range is continu accessed. This is accomplished by specifying the "–seek none" option. This is useful for testing the effectiveness of caching algorithms and/or the speed of the cache on the target device or file.

4.10 Reading and Writing Devices and Files

XDD read or writes devices or files. When running XDD on regular "files", the target files tend to be large and can "accidentally" be left behind after the testing. The –deletefile option will remove a targ file at the conclusion of the XDD run. This option is not recommended when running XDD on devic files for obvious reasons.

As previously implied, sequentially reading data is relatively straight forward: start at the beginning the device or file plus the starting offset and read data until the end of the data range is reached or the time limit expires. When reading a file it is assumed that the file is at least as large as the desired data range so that the read does not go past the end of the file. If the file is smaller than the desired or specified data range then a warning message is displayed regarding this condition.

Reading a file on most systems uses a file system buffer cache. In this mode, data is read into a file system buffer and then copied into the XDD I/O buffer by the CPU. This can cause two different problems with respect to the performance results. First, the maximum bandwidth for reading data int the machine for the first time is limited to the memory copy speed of the processor that can be much lower than the true bandwidth of the disk I/O subsystem under evaluation. Secondly, if all the data fr the file fits into the file system buffer then subsequent reads to the same file will be satisfied by copy the data from the file system buffer cache and not from the actual disk subsystem. In this case, XDD reporting the memory copy performance rather than the speed of the disk subsystem. To avoid this problem the Direct I/O, ("–dio"), option can be used to bypass the use of the file system buffer cache and forcing all read requests to access the disks. There are certain I/O request size and alignment restrictions that must be observed in order to use Direct I/O and these restrictions are Operating Syst dependent. These restrictions essentially state that the I/O requests must start on a "page-size" bound and must be in units of the native block size of the underlying file system.

Writing devices is very similar to read files except for the data transfer direction. However, writing "files" has additional complexities that can affect the performance results. This is due primarily to th allocation mechanisms used by the file system manager to allocate disk space when writing a file for first time. Generally speaking, each write operation will cause the file system manager to allocate spa on the disk to accommodate the data being written. These allocation operations can require additiona accesses to the disk that are "invisible" to XDD but do show up in the performance results as essenti slower write operations. In order to minimize the effects of this allocation process, the "–preallocate" option can be used to do the entire allocation operation before the file write operations take place. Th argument to this option specifies the number of blocks to pre-allocate before the write operations star This number should be greater than or equal to the number of blocks that will be in the file.

File write performance results can also be affected by the file system buffer cache. When writing a fi the data is first copied into the file system buffer and later actually written to the physical disk subsystem. As in the case of the read operation, it is possible to use the "–dio" option to avoid this da copy operation. The same request size and alignment restrictions apply Direct I/O in this case as in th read case.

One last note about writing to a target device or file. The data pattern to be written to the target can b specified by the "–pattern" option. This will fill the data buffer with a single character data pattern or with a random data pattern depending on the argument to this option. No data verification is done wi the data that is written out using this pattern. It is simply an option to use in case it is necessary to fin out where XDD has written to.

Finally, there is an option to re-align the internal data buffer within memory if necessary. This is use more for testing computer system memory performance rather than I/O performance of a target and therefore may have limited usefulness. It is only mentioned here for completeness.

4.11 Processor Allocation and Priority Assignment

On multi-processor systems it is possible to assign XDD threads to specific processors. This is accomplished with the –processor, –singleproc , and –roundrobin options. The –processor option all the explicit assignment of a processor to a specific XDD thread.

The –singleproc option will assign all XDD threads to a single processor specified as an argument to option. The –roundrobin option will distribute the XDD threads across M processors where M is the number of processors. M should be less than or equal to the number of processors in the system. The processor-numbering scheme used is 0 to N-1 where N is the number of processors in the system. Fo example, if there are five XDD threads running on a computer with eight processors, then the round robin processor assignment will assign threads 0 thru 4 on processors 0 thru 4. However, if there wer only two processors on the computer, then XDD threads 0, 2, and 4 will be assigned to processor 0 a threads 1 and 3 will be assigned to processor 1.

The priority of each thread defaults to the "normal" priority on the system. It is possible to increase t priority to a maximum level by using the "–maxpri" option. Maximizing the runtime priority of the XDD threads decreases the effects on the I/O performance of other processes running on the system. also possible to lock the XDD process and I/O buffers using the "–plock" option. This is done to prev the XDD process or any of its I/O buffers from being paged or swapped out of the system. The "–maxall" option is a shortcut for specifying both "–maxpri" and "–plock."

4.12 Preallocation Idiosyncrasies

Preallocation is used to tell the file system manager to allocate a specified amount of space in the file system before the actual write operations occur. The advantage to this is limiting the metadata overh involved with allocations during write operations but also to limit the number of file system "extents used to hold file data. Normally, when using DirectIO, each write operation can have the effect of

allocating a new file extent resulting in potential file fragmentation. The use of the "-preallocation" option circumvents this problem by allocating all the extents for the file ahead of time.
However, even though the space is allocated, it does not mean that the file size is that large. In fact, i possible to preallocate, say, 1GB of space and still have a zero length file.
Here is a synopsis of the various preallocation idiosyncrasies:

- Create a 0-length file
- Call xfsctl to reserve 16MB
- One 16MB extent is created for the file
- Write 8K into the file
- Close the file

The file is now 8KBytes long but takes up 16MB of physical space (one extent).

- Open the same file
- Call xfsctl to preallocate 64MB
- Close file

The file now takes up two extents:

- The first 16MB extent
- A second 48MB extent

The file is still 8KBytes in length and the 8KBytes are still valid but the file now occupies 64MB of space spanning two extents

- Open the file
- LSeek +32KBytes into the file
- Write 4KBytes
- Close file

The file is now 36KBytes in length

- The first 8 KBytes are valid
- Bytes 8K to 32K are NULL (as expected)
- Bytes 32K to 36K are valid

The file still retains its two extents (16MB+48MB)

- Open file
- LSeek +8KBytes into the file
- Write 3*8KBytes (three separate calls to write)
- Close file

The file is still 36KBytes

- The first 8K is still valid

- Bytes 8K-32K are new and correct
- Bytes 32K-36K are still valid

Note: All IO was done with DirectIO enabled. I also ran similar tests without DirectIO and the result were the same. Conclusion, DirectIO does not seem to affect the outcome because the space allocatio has already been done. With respect to allocation, the only difference between DirectIO and Bufferec is *when* the allocation is done. For DirectIO allocations are performed for each and every I/O in th order in which they are done. For Buffered IO the allocations are done at some later time based on th assumption that many adjacent blocks can be grouped into a single largish allocation rather than a bu of smallish ones as in the DirectIO case.

4.13 I/O Time Stamping

While running, each XDD thread has the option to enter time stamp information into a table that is la written to a file for further analysis. Each I/O operation is time stamped before the operation starts ar just after the operation ends. The time stamp table contains all the information necessary to understan when I/O operations started, ended, the block location being accessed, and the amount of data transferred. The time stamps themselves are taken from the system's high-resolution timer and re-normalized in units of picoseconds so that this data can easily be correlated with time stamp data fror the other associated XDD output files.

The format of the Time Stamp binary output file contains a header followed by the time stamp data.

In addition to a binary output file, the time stamp table information can be dumped in ASCII readabl text. There are several options for ASCII output including a summarized and a detailed output specif by the "–output summary" and "–output detailed" options. Appendix A includes examples of the summarized and detailed outputs of the Time Stamp table. The time stamp ASCII output file names have a ".csv" file extension so that it can more easily be read by a spreadsheet program such as Exce

5 Running XDD Programs

XDD programs include XDD itself, the timeserver program, and the gettime program. Example command line are given in Appendix B and are also contained in a file called tests.txt in the distribution bin directory.

5.1 XDD Command Line Arguments Synopsis

xdd
-blocksize *[target <target#>] number_of_bytes_per_block*
-bytes *[target <target#>] number_of_bytes_to_xfer_per_pass*
-combined *filename*
-createnewfiles *[target #]*
-csvout *filename*
-datapattern *[target <target#>]*
    character_pattern –or-
    “**random**” *–or-*
    “**sequenced**” *–or-*
    “**prefix**” *–or-*
    “**inverse**” *–or-*
    “**ascii** *<string>*” *–or-*
    “**hex** *<hex digits 0-9, a-f or A-F>*” *–or-*
    “**replicate**” *–or-*
    “**lfpat**” *–or-*
    “**ltpat**” *–or-*
    “**cjtpat**” *–or-*
    “**crpat**” *–or-*
    “**cspat**”
-deletefile *[target <target#>]*
-deskew
-dio *[target <target#>]*
-endtoend *[target <target#>]*
    issource | isdestination
    port *#*
    destination *hostname-or-IPaddress*
-errout *filename*
-flushwrite *#ops*

-fullhelp
-heartbeat *#seconds*
-help *option_name*
-id **commandline** - or - *“id_string”*
-kbytes *[target <target#>] number_of_kilobytes_to_transfer*
-lockstep *<master_target> <slave_target>*
    <when> <howlong>
    <what> <howmuch>
    <startup> <completion>
-maxall
-maxerrors *number_of_errors*
-maxerrorstoprint *number_of_errors_to_print*
-maxpri

-mbytes *[target <target#>] number_of_megabytes_to_transfer*
-memalign *[target <target#>] alignment_value_in_bytes*
-minall
-nobarrier
-nomemlock
-noproclock
-numreqs *[target <target#>] number_of_requests_to_perform*
-op *[target <target#>] read*|*write*
-output *filename*
-outputformat **add** | **new** *<format_id_string>*
-passdelay *seconds*
-passes *number_of_passes*
-passoffset *[target <target#>] offset_in_blocks*
-percentcpu **absolute | relative**
-preallocate *[target <target#>] number_of_bytes*
-processlock
-processor *processor_number target_number* -rwratio *[target <target#>] %read*
-queuedepth *[target <target#>] number_of_commands_per_target*
-qthreadinfo
-randomize *[target <target#>]*
-recreatefiles *[target #]*
-reopen *[target #]*
-reportthreshold *[target #] <#.#>*
-reqsize *[target <target#>] number_of_blocks*
-restart *[target <target#>]*
    enable
    frequency *<seconds>*
    file *<name_of_restart_file>*
    offset *<offset_in_bytes>*
-roundrobin #
-runtime *seconds*
-rwratio *[target #] <readwriteratio>*
-seek *[target <target#>]*
    save *filename*
    load *filename*
    disthist *#*
    seekhist *#*
    random
    range #
    stagger
    interleave #
    seed #
    none
-setup *setup_filename*
-sgio
-sharedmemory *[target <target#>]*
-singleproc *processor_number*
-startdelay *#seconds*
-startoffset *[target <target#>] starting_block_number*
-starttime *#seconds*

-starttrigger *targetA target>* ***time|op|percent|mbytes|kbytes*** *#*
-stoponerror
-stoptrigger *targetA target>* ***time|op|percent|mbytes|kbytes*** *#*
-syncio *number*
-syncwrite *[target <target#>] number*
-target *filename*
-targetdir *[target <target#>] directoryname pass*
-targetoffset *[target <target#>] offset_in_blocks*
-targets *N filename1 filename2 … filenameN*
-targetstartdelay *#seconds_multiplier*
-throttle *[target <target#>]*
    ops *operations/sec*
    bw *megabytes/second*
-timelimit *[target <target#>] seconds_per_pass*
-timeserver
    host *hostname*
    port *port#*
    bounce *bounce_count*
-timestamps *[target <target#>]*
    output *output_filename_prefix*
    summary
    detailed
    normalize
    summary
    wrap
    oneshot
    size #
    triggertime *#seconds*
    triggerop *operation#*
    append
    dump *dump_filename_prefix*
-verbose
-verify *[target <target#>]*
    location
    contents
-version

5.2 Detailed Option Specifications

-blocksize specifies the number of bytes per block. This defaults to 1024 bytes per block. Block sizes must be powers of 2 or the results are unpredictable.

-bytes specifies the number of bytes to transfer per pass. This can be any positive number up to 2^{64}-1 See also: -**kbytes** and **-mbytes**.

-combined *filename* will append just the "Combined" results output to the file specified by *filename*. This allows for collecting the Combined performance data from multiple runs in a single file. This do not include error messages. See the **–errout** option for more information on redirecting error output.

-createnewfiles will cause new target files to be created for each pass in an XDD run. Each new file the file name that was given as the target file name but it is appended with a number that represents t pass in which it was created. It is not a good idea to run this on a target "device" file because the new target will get recreated as regular files and not a special device files.

-csvout *filename* will send all the results output to the file specified by *filename* similar to the –outpu option. The difference between -csv output and the normal output is that this is a Comma Separated Values file that is directly importable into MS Excel. This does not include error messages. See the –**errout** option for more information on redirecting error output.

-datapattern specifies either a single byte data pattern character or a special operator as described below. (*Note: This option was formerly called –pattern*)

- If the operator "**random**" is specified then a random data pattern is generated in the entire I/O buffer.
- If the operator "**sequenced**" is specified, then the data pattern will be sequential 64-bit intege starting with the current block address times the size of a 64-bit integer. It should be noted tha writing this sequenced pattern to the device will result in additional CPU overhead that may affect overall performance. Similarly, for read operations, if the "**sequenced**" data pattern is specified, the data is checked to see if the data read is in fact what was expected.
- If the word "**prefix**" is specified for the pattern then the specified hex digits will be placed in upper N bits of the 64-bit pattern
- If the operator "**inverse**" is specified for the pattern then the actual pattern will be the 1's compliment of the specified pattern
- If the word "**ascii**" is used as the data pattern, then the data pattern will be the string specified after the "**ascii**" operator. The string is only written once at the beginning of the I/O buffer an not repeatedly copied throughout the I/O buffer.
- If the word **"hex"** is specified then the following hex characters <0-9,a-f or A-F> are used as pattern
- If the word "**replicate**" is specified then whatever pattern was specified is replicated througho the buffer
- If the word "**lfpat**" is specified then the low-frequency 8B/10B pattern is used
- If the word "**ltpat**" is specified then the low-transition 8B/10B pattern is used
- If the word "**cjtpat**" is specified then the compliant jitter 8B/10B pattern is used
- If the word "**crpat**" is specified then the compliant random 8B/10B pattern is used
- If the word "**cspat**" is specified then the compliant sequential 8B/10B pattern is used
- The default data pattern is all binary 0's

-deletefile will cause the target file to be deleted at the end of a run. To have files deleted and recreat between passes, see the **–recreatefiles** option.

-deskew will adjust the performance calculations to deskew the results. See section on Deskew for m information.

-dio will turn on the DIRECT IO option when accessing a regular file. This option cannot be used wh accessing special device files. Certain blocksize, request size, and alignment restrictions apply and w cause problems if the wrong combination of block size, request size, and offsets are chosen.

-endtoend specifies that the target device will participate in an End-to-End operation. There are two sides to an End-to-End (aka e2e) operation: the *source* and the *destination*. It takes two concurrent instances of *XDD* to perform an e2e operation: one running on the *source* side and one running on th *destination* side. The instance of *XDD* on the *source* side of the e2e operation will read the specified target file/device and transfer the data over a TCP/IP socket to the instance of *XDD* on the *destinatio* side. The *destination* instance of *XDD* will receive the incoming data and write it to its specified targ file/device. See section on Examples for more information.
The e2e option has the following operators:

- If the operator "**issource**" is specified then this particular instance of *XDD* assumes that it is o the *source* side of an e2e operation.
- If the operator "**isdestination**" is specified then this particular instance of *XDD* assumes that on the *destination* side of an e2e operation.
- The operator "**destination** *hostname*" specifies the name of the host on the *destination* side of e2e operation. The *hostname* can be specified as a name or an IP Address. It is necessary to specify this address on both instances of *XDD*, *source* side <u>and</u> *destination* side, in order to explicitly define the correct network interface to use for the data transfer.
- The "**port** #" defines the specific port to use for the data transfer. In the event that multiple threads are used on each side of the e2e operation (i.e. –queuedepth 48) then the port number becomes the base port number and is incremented by 1 for each subsequent qthread. For example, if "–queuedepth 4" is specified along with "–e2e port 3010" then the port assignmen will consume port numbers 3010, 3011, 3012, and 3013 for qthreads 0, 1, 2, and 3 respectivel For this type of operation it is important to choose a base port number that has free ports with the range of ports that will be consumed. The default port number at the time of this writing i 5044 decimal.

-errout *filename* will send all the error message output to the file specified by *filename*. This does no include normal results output. See the **–output** option for more information on redirecting results out

-flushwrite will force a sync() operation to occur every so many write operations as specified by the associated argument to this option.

-fullhelp will display a list of these options *and* a one line explanation of each.

-heartbeat *#seconds* will display the current operation and pass number for each target I/O thread ev N seconds where N is specified as *#seconds*.

-id allows the user to specify an ASCII string to be displayed in the output in order to identify the run the word **commandline** is used as the character string then the input command line is used as the id. Multiple instances of the -id option will concatenate each specified id to the previous one.

-kbytes specifies the integer number of kilobytes to transfer on each pass. In this case, one kilobyte i
equal to 1024 bytes. If the –**numreqs** option is also specified, –**numreqs** takes precedence.
See also: **-mbytes** and **-bytes**.

-lockstep *<master_target> <slave_target> <when> <howlong> <what> <howmuch> <startup> <completion>*
Where

"master_target" is the target that tells the slave when to do something.
"slave_target" is the target that responds to requests from the master.
"when" specifies when the master should tell the slave to do something.
The word "***when***" should be replaced with the word:
"**time**"
"**op**"
"**percent**"
"**mbytes**"
"**kbytes**"

"howlong" is either the number of seconds, number of operations, ...etc.

- The interval time in seconds (a floating point number) between task requests from the master to the slave. i.e. if this number were 2.3 then at every 2.3-second interval the master would request the slave to perform its task.
- The operation number that defines the interval on which the master will request the sl to perform its task. i.e. if the operation number is set to 8 then upon completion of eve 8 (master) operations, the master will request the slave to perform its task.
- The percentage of operations that must be completed by the master before requesting slave to perform a task
- The number of megabytes (1024*1024 bytes) or the number of kilobytes (1024 bytes

"what" is the type of task the slave should perform each time it is requested to perform a task by the master. The word "what" should be replaced by:
"**time**"
"**op**"
"**mbytes**"
"**kbytes**"

"howmuch" is either the number of seconds, number of operations, ...etc.

- The amount of time in seconds (a floating point number) the slave should run before pausing and waiting for further requests from the master.
- The number of operations the slave should perform before pausing and waiting for further requests from the master.
- The number of megabytes (1024*1024 bytes) or the number of kilobytes (1024 bytes) slave should transfer before pausing and waiting for further requests from the master

"startup" is either "**wait**" or "**run**" depending on whether the slave should start running upon invocation and perform a single task or if it should simply wait for the master to request it to perform its first task.

"completion" - in the event that the master finishes before the slave, then the slave will have the option to complete all of its remaining operations or to just stop at this point. This should be specified as either "**complete**" or "**stop**".

-maxall will set the -maxpri and -plock options.

-maxerrors *number_of_errors* will limit the number of errors to *number_of_errors* so as not to clutt up the output with endless lines of errors. Once this limit has been reached the XDD pass will end.

-maxerrorstoprint *number_of_errors_to_print* will limit the number of errors that actually get displayed to *number_of_errors_to_print* so as not to clutter up the output with endless lines of errors Once this limit has been reached processing will continue but further errors will be accumulated but n displayed.

-maxpri will set the priority of all XDD threads to maximum. NOTE: Use of this option can result in system hangs due to deadlocks with other system functions.

-mbytes specifies the integer number of megabytes to transfer on each pass. In this case, one megaby is equal to 1024*1024 or 1048576 bytes.
If the –**numreqs** option is also specified, –**numreqs** takes precedence. See also: -**kbytes** and **-bytes**.

-memalign *memory_alignment_value_in_bytes* will cause the internal memory address alignment of I/O buffer to be offset by the number of bytes specified as *memory_alignment_value_in_bytes.* The I buffer is normally page aligned.

-minall will set the –noproclock and -nomemlock options

-nobarrier will cause the passes to run asynchronously.

-nomemlock will prevent the XDD memory buffers from being locked so that they can be paged or swapped out.

-noproclock will prevent XDD process from being locked in memory so that it can be paged or swapped out.

-numreqs specifies the integer number of "reqsize" requests to perform on each pass.
If the –**mbytes** or –**kbytes** option is also specified, –**numreqs** takes precedence.

-op specifies the operation to perform: either **read** or **write** may be specified.

-output *filename* will send all the results output to the file specified by *filename*. This does not inclu error messages. See the **–errout** option for more information on redirecting error output.

-outputformat will either "add" items to the XDD output lines or create a "new" output line. See section on Output Format for more information.

-passdelay # where # is the number seconds to delay between passes.

-passes # where # the number of passes to perform.

-passoffset # where # is the number of blocks to offset for each pass.

-preallocate # will preallocate # bytes. This is used when writing to a target that is a regular file. Thi option has no effect when reading or when the target is a real device. See section on "Preallocation Idiosyncrasies" for more details.

-processor *procesor_number target_number* This option allows an *XDD* thread for a specific target run on a specific processor. The *XDD* thread for target *target_number* is assigned to processor *processor_number.*

-processlock will lock the XDD process in memory so that it cannot be paged or swapped out. This i useful on a crowded system.

-queuedepth specifies the number of commands to send to each target at one time. This exercises th command queuing capabilities of a storage device or, if I/O if to a file, it will mimic parallel I/O – multiple readers/writers to a single file.

-randomize will cause the seek locations to be randomized between passes.

-recreatefiles will cause the target files to be closed, deleted, and re-created for each pass in an XDD run. It is not a good idea to run this on a target "device" file because the target will get recreated as a regular file and not a special device file.

-**reopen** will cause the target file to be closed and re-opened for each pass in an XDD run.

-**report_threshold** will report the byte location of the operation that exceeded the specified threshold time.

-reqsize specifies the number of *blocks* to transfer where the size of the block is specified by the – blocksize parameter.

-restart indicates that a previously failed end_to_end operation failed and must be resumed at some point into the file being transferred from a source to a destination (see also –**endtoend**) . The option parameters are:

- **enable** – requires no parameters but tells the end_to_end operation to start the restart_manage that will keep track of the most recent successful write operation on the destination side of an end_to_end operation. The restart information is written to a restart file.
- **frequency** *seconds* – specifies the number of seconds between updates to the restart file.
- **file** *filename* – specifies the name of the restart file to generate during an end_to_end operatic
- **offset** *#bytes* – specifies the number of bytes into the source/destination file to use as the poir which to resume the data copy.

-roundrobin # will assign successive XDD threads to processors in a "roundrobin" fashion across # processors.

-runtime #seconds will cause XDD to terminate completely after it has run for the specified number seconds. It is important to note that if the *timestamp* option is also specified the timestamp buffer *wra* option is automatically enabled so that the internal timestamp buffer is not overrun. .

-rwratio (or **–rw)** specifies the percentage of operations that should be read operations. The remainir operations will be write operations. For example, specifying a value of 30.2 (i.e. ***–rwatio*** *30.2*) will cause 30.2% of the total number of operations being performed on the target to be *read* operations ar 69.8% of the operations will be write operations. Values less than 0 or greater than 100 are not allow

-seek specifies a number of parameters that are specific to the access pattern used on each target. The default access pattern is purely sequential. These parameters are:

- **save** *filename* - will save the list of seek locations in an ASCII text file specified by *filename.* This file can later be used by the –**seek load** option. See Appendix B for the format of this fi
- **load** *filename* - will load the list of seek locations from an ASCII text file specified by *filenar*
- **range** #blocks - will specify the range in blocks over which to perform random seek operatio
- **random** will generate a random list of locations to access over the “range”
- **seed** *seed_value* specifies a seed value to use when generating random locations
- **stagger** will stagger the requests sequentially and evenly over the “range”
- **interleave** *factor* where “factor” is the interleave factor to used (see section on parallel I/O ar seek interleave).
- **none** will cause XDD to continuously read the *starting* block on a target until for a total of –mbytes or –numreqs of data transfers completes.
- **disthist** *#categories* – will display an ASCII readable histogram of the seek “distances”.
- **seekhist** *#categories* – will display an ASCII readable histogram of the seek “locations”.

-setup specifies a file that contains commonly used XDD options. This file is read in and the options contained within the file will be inserted into the command line.

-sgio will perform I/O operations to the specified target using the SCSI Generic protocol rather than t normal read/write system calls. This is only valid on Linux systems and is used to provide raw-like access to a device.

-sharedmemory tells XDD to use a shared memory segment (via shmget/shmat) for the I/O buffer rather than using the normal valloc()/malloc() system calls.

-singleproc *processor_number* will assign all XDD threads to the specified processor.

-startdelay *#seconds* will cause the target I/O threads to all start after a specific startup delay specifi in seconds.

-startoffset specifies the starting block number. This defaults to block 0. The value must be a positiv integer.

-starttime *global_time* will cause the target I/O threads to all start at the specified time. The global ti is the time value returned by the time server and is consistent for all systems using the time server. Se the **–timeserver** for more information.

-syncio *number* will cause each of the XDD I/O threads to synchronize every nth I/O operation whe is specified as “*number”*. .

-syncwrite will cause each of the XDD I/O threads to synchronize write operations at the end of eacl pass flushing all data to the physical media.

-targetdir specifies the name of the directory to be pre-pended to the target(s). For example, specifyi a parent directory of */dev/rdsk/* (i.e. ***–targetdir*** */dev/rdsk/*) and target names of “*dks1d2s0 dks7d3s0*” will cause I/O to be directed to */dev/rdsk/dks1d2s0* and */dev/rdsk/dks7d3s0* respectively. It is importa to remember to put the trailing slash (“/”) at the end of the parent directory name.

-targetoffset specifies the offset in blocks that is used by each XDD process to determine their respective starting locations. The purpose of this is to be able to run multiple XDD threads on a singl device but to have each thread start at a different location. (*Note: This option was formerly called –procoffset*)

-targets must first specify the number of targets (*N*) followed by the target device names or file name of each of the *N* targets. For example, "***-targets** 2 dks1d2s0 dks7d3s*" will perform I/O to the targ devices *dks1d2s0* and *dks7d3s0* respectively. In the output reports, these two targets will also be identified as targets 0 and 1 respectively.

-targetstartdelay *#.#seconds* will cause each target I/O thread to all start after the specified number seconds has elapsed from when the previous target started. For example, a value of
"-targetstartdelay 1.2"
would allow target 0 to start immediately, target 1 to start 1.2 seconds after target 0, target 2 to start 1 seconds after target 1 and so on.

-throttle specifies the I/O Operations per second (**ops**) or bandwidth (**bw**) limit for the target(s) depending on which of the two parameters are specified. Valid values are positive real numbers great than 0.000. The parameters **ops** and **bw** are mutually exclusive and the last one specified for a target takes precedence. Example usages:

- *"-throttle ops 7.8"* will limit all targets to running at 7.8 I/O operations per second.
- *"-throttle bw 87.2"* will limit all targets to running at 87.2 megabytes per second.
- *"-throttle target 2 ops 7.8"* will limit only target 2 to running at 7.8 I/O operations per secon and all other targets will have no throttle limit unless specified with another *–throttle* option.

-timelimit # will impose a time limit of # seconds on each pass. This value must be a positive integer

-timeserver is used to specify the hostname of the time server that acts as the 'master clock' for all timing information when running XDD across multiple machines. This option takes one of three operators as described below. These operators take the place of previous options such as –port and –bounce. See the **–starttime** option for additional information.

- If the operator "**host**" is specified then the specified *hostname* is used as the time server.
- If the operator "**port**" is specified then the specified *port_number* is used to connect to the ti server.
- If the operator "**bounce**" is specified then the specified *bounce_count* specifies the number of times to access the time server in order to resolve the time delta between the time server and t client.

-ts or **-timestamps** specifies a number of parameters that are specific to the time stamping capabiliti These are:

- **summary** will generate a summary of all the I/O operations in the time stamp trace (see figur for details).
- **detailed** will generate a detailed report of each I/O operation in the time stamp trace and a summary report. It is recommended that the **output** filename be specified when using **detaile** reporting since the trace data can be exceedingly verbose (see figure 4 for details).

- **normalize** will cause all the time stamp values to be normalized to the global clock. This is useful when running XDD on multiple machines so that the events in the time stamp file can correlated in time.
- **output** *output_filename* will cause the detailed and/or summary reports to be written to a file *"output_filename"*. The output defaults to standard out.
- **append** will cause the detailed and/or summary reports to be appended to the specified outpu file.
- **dump** *dump_filename_prefix* will dump a binary file that contains all the time stamp data. Th following structure contains the format of that file.
- **wrap** will cause the internal timestamp buffer to *wrap* around to the beginning of the buffer if/when the end is reached. This is used in conjunction with the "size" option described below The reason for wrapping the timestamp buffer is to essentially capture the most recent I/O operations in a timestamp buffer that is smaller than required for the number of operations be processed by XDD.
- **oneshot** specifies that time stamping will stop once the internal timestamp buffer is full.
- **size #** specifies the size of the internal timestamp buffer in terms of the number of operations will fit into the buffer. If the size specified is smaller than the number of operations to be performed, the timestamp buffer will be "*wrapped*" after the last timestamp buffer entry is us
- **triggertime #***seconds* will cause timestamping to start at the specified time as measured in *global-time seconds*.
- **triggerop** *operation#* will cause timestamping to start when the specified operation number i reached.

-verbose will display performance information for each pass.

-verify specifies a number of parameters that are specific to the time stamping capabilities. These are

- **location** will verify that the location that was just read is the intended location. This options makes the assumption that the storage device was previously written with a "sequenced" data pattern (see **–datapattern** ***sequenced*** option).
- **contents** will compare the data read with the specified data pattern. It is required that the data patterns exist on the disk before this option is used otherwise the contents cannot be compare anything reasonable. Furthermore, it is recommended that any data pattern other than "rando be used for the data compare operation (see **–datapattern** option for detailed on specifying d patterns).

-version will display the version number for this XDD program.

5.3 Target-specific options

Many of the options can be target-specific. These options are listed with the optional ***[target*** *<target#]* argum that immediately follow the option name. The word "**target**" indicates that the associated option is to be set fo the target with a target number of <*target#*>. Target numbers are from 0 to N-1 where N is the number of targ being accessed in this run. For example, specifying "***–op target 3 read"*** will cause target 3 to issue read operations regardless of what the other targets are doing. This capability is useful for tailoring the behavior of each target in a run to meet specific I/O requirements. For example, it is possible to have a single XDD run accessing several targets using different throttle values so that one target does not overwhelm the others.

It is important to note that the options are evaluated from left to right on the command line or from top to bott in a setup file and that latter options (to the right) take precedence over prior options (to the left). Take the following command line for example:

```
XDD –op read –op target 1 write –targets 3 s1 s2 s3 –reqsize 1 –numreqs 7
```

The "**–op read**" option will cause all three targets (s1, s2, and s3) to perform *read* operations. However, the " **op target 1 write**" option will override the *read* operation for target number 1 (target s2) causing it to perform *write* operations.

5.4 Lockstep Operations

Lockstep operations are used to simulate the I/O interaction between multiple applications running o system. For example, one application may be creating files that a second application will use just afte their creation – aka the "read-after-write" scenario where a file is being ingested from a source, writt to a file and another application is reading blocks just after they are written in order to process the da as quickly as possible.
An example of this is as follows:

```
XDD -targets 2 /dev/disk1 /dev/disk1 \
   -op target 0 write \
   -op target 1 read \
   -reqsize 1024 -mbytes 2048 \
   -lockstep 0 1 op 1 op 1 wait complete
```

This will cause target 0 to write a block and then signal target 1 to start reading. Since they are the sa target starting at the same locations, target 1 will be exactly 1 operation behind target 0 all the time. Essentially this tells target 0 to do 1 operation, signal target 1 which will do 1 operation and then wai for target 0 to signal it again and so on.

The current version of XDD only supports lockstep operations on a single computer system. The nex version of XDD will enable lockstep operations across physically separate computer systems.

5.5 Timeserver and gettime

Command synopsis of the timeserver command is

timeserrver [-**port** *#*]

Where

-port # specifies the port number to use for the time serving function.

The gettime command is used to obtain the global clock value from the timeserver computer. The command synopsis is:

gettime
-timeserver *hostname*
-port *#*
-add *seconds*
-bounce *times*
-verbose
-waitfortime *milliseconds*

Where:

-timeserver *hostname* specifies the name of the computer running the timeserver. This may be eithe host name or an IP address.

-port # specifies the port number to use when contacting the timeserver.

-add *seconds* specifies number of seconds to add to the global time that is displayed as the output of program.

-bounce *times* specifies the number of times to ping the timeserver in order to get a minimum round trip time. The higher the bounce count, the more accurate the global time will be.

-verbose will cause gettime to display more information than simply the global clock value.

-waitfortime will cause gettime to wait until the specified global time in milliseconds is reached at which time gettime will complete. This is useful when running it in a script to block the execution of script until a particular global time.

5.6 Deskew

De-skewing the performance results becomes particularly necessary when testing a large number of targets on a single system. The reason is that when all devices are started there can be a significant amount of time lag between the time the first targets starts and the last target starts its data transfer. Furthermore, there may be a long lag time between the time the first target finishes and the last targe finishes particularly if there is one device that is unusually slow. This causes the overall results to be skewed and does not represent the true “cross sectional” bandwidth of the system as a whole.

The deskew option will report the bandwidth during the time in which all the targets are transferring data. This is effectively from the time the last target starts to the time the first target finishes. During time period all targets are transferring data. This “deskewed” bandwidth is a more accurate representation of the bandwidth of the system.

6 Runtime Hints

6.1 Windows

The Windows physical drive is the equivalent of the raw volume in a Unix environment. The physical drive ca
be specified by using the following XDD command line:

```
XDD –op read –targets 1 \\.\\PhysicalDrive0 –reqsize 1 –mbytes 1
```

XDD may complain about an incorrect function but that message can be ignored. It will be fixed in a future release. If a *cygwin* shell window is being used then the syntax must include extra "\" characters like so:

```
XDD –op read –targets 1 \\\\.\\\\PhysicalDrive0 –reqsize 1 –mbytes 1
```

6.2 Linux

Raw I/O is supported using either the SCSI Generic devices (i.e. /dev/sgX where "X" is the device number) or
"raw" command to create the "raw" device associated with a block device. For the best "raw-like" performanc
the SCSI Generic devices should be used.

The Linux "raw" devices are only an approximation of real raw devices on most other Unix systems. It should
also be mentioned that request sizes larger than about 256Kbytes may not actually be issued to the device but
rather be split into multiple requests by the "raw" device driver.

6.3 FreeBSD

6.4 OSX

6.5 Solaris

There is an issue with running XDD on Solaris that involves the default number of semaphores that Solaris all
each process to use. It is normally too low for XDD to run and XDD generates errors that say things like "can
allocate barrier for …, not enough space". The following parameters can be put in /etc/system on a solaris syst
and a reboot will take care of this problem:

```
set semsys:seminfo_semmni=4096
set semsys:seminfo_semmns=8192
set semsys:seminfo_semmap=4098
set shmsys:shminfo_shmmni=512
set shmsys:shminfo_shmseg=32
```

Another change that is sometimes necessary is

```
set maxphys = 0x1000000
```

This will change the maximum request size to 16 MB that is useful when doing really large request sizes for ra
device tests.These settings may be a little high and can be adjusted to meet the local system constraints.

6.6 IRIX

IRIX is no longer supported but was reasonably well behaved and had no known idiosyncrasies.

6.7 AIX

AIX can use very large page sizes (on the order of 16Mbytes per page) but it is still necessary to “pin” all the pages in memory for every I/O operation. For most I/O performance testing this is not a problem unless the number of targets gets large and/or the bandwidth is very high. In this case the page pinning operation takes a significant percentage of the I/O time and can have a negative affect on the results.

To avoid the page pinning penalty, the “-sharedmemory” operation can be used to allocate the I/O buffer in a shared memory segment which causes the system to bypass the page pining since shared memory pages are already pinned by default.

7 Output and Reports

7.1 Reporting Options

The –verbose option will display performance information for each pass within an XDD run along w
per-target totals and combined averages. There is also an "id" option (–id) that will display a specifie
string of identification information along with the normal output of the run. If the id string is specifie
be "commandline" then the entire XDD command line plus all the options will be display. This is use
when running XDD from a shell script to get more qualitative information about the run into the outp
file.

Since XDD I/O operations can fail due to device failures or option specifications it is possible to generate tremendous numbers of error messages. To avoid filling up output files with too many error
messages, the –maxerrors option can be used to limit the number of error messages to display to som
small, finite number. Once this number of errors has been reached, I/O operations to that target are halted for that pass.

7.2 Output Format

The "–outputformat" option allows the user to specify which variables should be displayed in the output line.
example, given a normal XDD command line with the usual options, it is possible to use the "-outputformat" option as follows:

```
      XDD …  -outputformat   "+PASS+TARGET+BYTESREAD+OPS+READBANDWIDTH"
```

To produce output that would only contain the pass number, target number, bytes "read", operations performe
and achieved "read" bandwidth.

The following is a list of defined output format identifier strings that can be used. Each format identifier string
must begin with a "+" as indicated.

+WHAT
+PASS
+TARGET
+QUEUE
+BYTESTRANSFERED
+BYTESXFERED
+BYTESREAD
+BYTESWRITTEN
+OPS
+READOPS
+WRITEOPS
+BANDWIDTH
+READBANDWIDTH
+WRITEBANDWIDTH
+IOPS
+READIOPS
+WRITEIOPS
+LATENCY
+ELAPSEDTIME1STOP
+ELAPSEDTIMEPASS
+OVERHEADTIME
+PATTERNFILLTIME
+BUFFERFLUSHTIME

+CPUTIME
+PERCENTCPUTIME
+PERCENTCPU
+USERTIME
+USER
+USRTIME
+SYSTEMTIME
+SYSTEM
+SYSTIME
+PERCENTUSER
+PERCENTUSERTIME
+PERCENTUSR
+PERCENTSYSTEM
+PERCENTSYSTEMTIME
+PERCENTSYS
+OPTYPE
+XFERSIZEBYTES
+XFERSIZEBLOCKS
+XFERSIZEKBYTES
+XFERSIZEMBYTES
+E2EIOTIME
+E2ESRTIME
+E2EPERCENTSRTIME

+E2ELAGTIME
+E2EPERCENTLAGTIME
+E2EFIRSTREADTIME
+E2ELASTWRITETIME
+DELIM

** Aside from the obvious o
the "+DELIM" can be used
print a specific delimiter
between variables that get
displayed. The "+WHAT" is
used to indicate which outpu
line is being displayed: PAS
QThread Average, Target
Average, Combined, …etc.
the others are reasonably sel
explanatory

The output of XDD comes in several sections.

- The first section describes the state of options selected for the run that apply to all the targets being tested.
- The second section contains information about the system that XDD is being run on.
- The third section of output describes the options that apply to each of the individual targets.
- **The fourth section describes the rate information.**

```
root# ./xdd -op write -targets 1 /space/testfile -reqsize 4096 -mbytes 2048 -dio -queuedepth 4 -passes 3 -verbose
target_bytes_to_xfer_per_pass is 2147483648, iosize is 4194304, target_ops is 512
IOIOIOIOIOIOIOIOIOIOI XDD version Linux.7.1.0.rc1.052510.Build.1030 based on Linux.7.1.0.rc1.050810.Build.2222 IOIOIOIOIOIOIOIOIOIOI
xdd - I/O Performance Inc., US DoE/DoD Extreme Scale Systems Center <ESSC> at Oak Ridge National Labs <ORNL> - Copyright 1992-2010

XDD DISCLAIMER:
 *** >>>> WARNING <<<<
 *** THIS PROGRAM CAN DESTROY DATA
 *** USE AT YOUR OWN RISK
 *** IOPERFORMANCE and/or THE AUTHORS ARE NOT LIABLE FOR
 *** >>>> ANYTHING BAD <<<<
 **** THAT HAPPENS WHEN YOU RUN THIS PROGRAM
     ...although we will take credit for anything good that happens
     but we are not *liable* for that either.

Starting time for this run, Tue May 25 10:33:59 2010

ID for this run, 'No ID Specified'
Maximum Process Priority, disabled
Passes, 3
Pass Delay in seconds, 0
Maximum Error Threshold, 0
Target Offset, 0
I/O Synchronization, 0
Total run-time limit in seconds, 0
Heartbeat 0
Output file name, stdout
CSV output file name,
Error output file name, stderr
Pass synchronization barriers, enabled
Number of Targets, 1
Number of I/O Threads, 5

Computer Name, darla, User Name, tmruwart
OS release and version, Linux 2.6.31.5-0.1-desktop #1 SMP PREEMPT 2009-10-26 15:49:03 +0100
Machine hardware type, x86_64
Number of processors on this system, 2
Page size in bytes, 4096
Number of physical pages, 2047554
Megabytes of physical memory, 7998
Clock Ticks per second, 100
Seconds before starting, 0
./xdd: xdd_target_existence_check: NOTICE: target number 0 name /space/testfile does not exist so it will be created.
./xdd: xdd_target_existence_check: WARNING: Target number 0 name /space/testfile must be a regular file when used with the -dio flag
```

```
                Target directory, "./"
                Process ID, 12911
                Thread ID, 12912
                Processor, all/any
                Read/write ratio,  0.00 READ, 100.00 WRITE
                Throttle in MB/sec,   0.00
                Per-pass time limit in seconds, 0
                Pass seek randomization, disabled
                File write synchronization, disabled
                Blocksize in bytes, 1024
                Request size, 4096, 1024-byte blocks, 4194304, bytes
                Number of Operations, 512
                Total data transfer for this TARGET,
                        2147483648, 1024-byte Blocks
                             2147483648,     Bytes
                                2097152.000, KBytes
                                   2048.000, MBytes
                                      2.000, GBytes
                                      0.002, TBytes
                Start offset,   0
                Pass offset,   0
                Seek Range,
                        1073741824, 1024-byte Blocks
                             1073741824,     Bytes
                                1048576.000, KBytes
                                   1024.000, MBytes
                                      1.000, GBytes
                                      0.001, TBytes
                Seek pattern, sequential
                Flushwrite interval, 0
                I/O memory buffer is a normal memory buffer
                I/O memory buffer alignment in bytes, 4096
                Data pattern in buffer,0x00
                Data buffer verification is disabled.
                Direct I/O, enabled
                Preallocation, 0
                Queue Depth, 4
                Timestamping, disabled
                Delete file, disabled

----------------------All targets should start now------------------------
What              Pass  Target   Queue     Bytes_Xfered        Ops     Elapsed  Bandwidth       IOPS   Latency    Pct_CPU  Op_Type     Xfer_Size
    UNITS>>     Number  Number  Number            Bytes       #ops     seconds   MBytes/s      Ops/s  millisec    percent     text         bytes
 TARGET_PASS         1       0       4       2147483648        512      17.398    123.436     29.429    33.980      2.357    write       4194304
 TARGET_PASS         2       0       4       2147483648        512      18.403    116.690     27.821    35.944      2.337    write       4194304
 TARGET_PASS         3       0       4       2147483648        512      18.345    117.060     27.909    35.830      2.289    write       4194304
 TARGET_AVERAGE      3       0       4       6442450944       1536      54.146    118.983     28.368     8.813      2.327    write       4194304
 COMBINED            3       1       4       6442450944       1536      54.146    118.982     28.368     8.813      2.327    write       4194304
Ending time for this run, Tue May 25 10:34:53 2010
 This run terminated normally
root#
```

The default output of XDD is a line of text with the following format

```
 What     Pass  Target  Queue  Bytes_Xfered   Ops  Elapsed  Bandwidth  IOPS   Latency  Pct_CPU  Op_Type  Xfer_Size
UNITS>> Number Number  Number   Bytes        #ops  seconds  MBytes/s  Ops/s  millisec  percent  text      bytes
```

The first line indicates the variable being reported, the second line provides the *UNITS* for each variable, and the lines beyond that are the variables themselves.
The fields have the following meanings:
WHAT This is an identifier that explains how the results should be interpreted for the given line. The possible values of "WHAT" are:

- TARGET PASS – displayed only when the –verbose option is specified. Provides the results for a specific target for a particular pass. The target results are a combined average of all the associated qthreads for the target. The "Q" result in this case reflects the total number of qthreads operating on behalf of the target.
- QUEUE PASS – displayed only when the –qthreadinfo option is specified. Provides the results for a specific qthread for a particular pass. The "Q" result in this case reflects the specific qthread number relative to zero for this target. For example, if there are 4 qthreads (i.e. –queuedepth 4) then the Q values will range from 0 to 3.
- TARGET AVERAGE – displayed only when the –verbose option is specified. Provides the results for a specific target averaged over all passes in the run.
- COMBINED – Provides the combined results over all targets and qthreads for all passes in a run.

PASS is the target pass number of the specific result. Pass numbers start at 1.
Target is the target number, relative to 0 (zero). A list of the targets and the associated numbers precedes this line of output.
Queue – see above.
Bytes_Xfered Is the total number of bytes that were transferred during the XDD run for this target.
Ops is the total number of read or write operations performed on this target.
Elapsed is the number of seconds that elapsed for the current pass in a multi-pass run for the target or qthread being reported.
Bandwidth is the average I/O rate in Mega Bytes per Second where 1 MByte/sec equally 10^6 bytes per second.
IOPS is the Average number of I/O operations per second for this pass in a multi-pas run.
Latency is the Average time it takes to perform each operation.
Pct_CPU is the percentage of the CPU that was used by all the qthreads on behalf of this target. This includes system and user time.
Op_type is either read, write, or mixed.
Xfer_Size is the *average* request size in block-size blocks that was used for that target or qthread. Generally this is a constant value for any particular test run.

If the **–verbose** option is specified then each line has a pass number associated with it and the final output lines report overall averaged values for the Average Rate and Elapsed time. For a Multi-Target run, the sum total of all the targets is presented as the **Combined** average. For example, if *two* targets are being tested and *each* target performs at an average of 75MB/sec, the Combined average is 150 MB/sec.

7.3 What the numbers really mean

There are several performance values reported. These include the per-pass target results, the individual queue results, the target averages over all passes, and the combined average of all targets over all passes.

8 Performance Tuning Hints

This section describes various hints about performance tuning.

8.1 Caches and write performance

When writing to a storage device whether it is a single disk drive or a disk array it is important to know the sta
of the caches on each of the devices that have cache. For maximum performance for write operations, it is necessary to enable the write caches on a disk array controller as well as the write caches on the disk drives themselves.

8.2 Fibre Channel Frame Sizes

Fibre Channel host bus adapters (HBA), switches, and target devices all have frame sizes defined and negotiat
when any two FC devices are connected together. For maximum bandwidth performance it is important to ma
certain that the FC frame size is set to 2048-bytes. For maximum transaction performance for small transactio
sizes (i.e. around 512 bytes per transaction) a smaller frame size of 512 or 1024 bytes can be used.

9 Examples

The following is a list of examples on how to run XDD.

9.1 Example 1 – Basic XDD command line

```
XDD –op read –targets 1 /dev/rdsk/dsk10d2s0 –reqsize 128
-mbytes 64 –passes 3 –verbose
```

This is a very basic test that will **read** sequentially from target device **/dev/rdsk/dks10d2s0** starting
block 0 using a fixed request size of **128** blocks until it has read **64** MegaBytes (64 * 1024*1024 byt
It will do this **3** times and display performance information for each pass. The default block size is 1
bytes per block so the request size in bytes is 128 Kbytes (128 * 1024 bytes). Please note that all thes
options need to be on a single command line unless they are in the setup file where they can be on
separate lines.

9.2 Example 2 – Specifying multiple targets and timelimit

```
XDD –op write –targets 2 /raid/BIGFILE1 /raid/BIGFILE2
-blocksize 512 –reqsize 128 -mbytes 64 –verbose
–passes 3 -timelimit 10
```

This test will **write** sequentially from **2** target files **`/raid/BIGFILE1`** and **`/raid/BIGFILE2`**
starting at the beginning of each file using a fixed request size of **128** blocks of **512** bytes per block u
it has read **64** MegaBytes (64 * 1024*1024 bytes) – *or* – until it has reached a time limit of **10** secon
which time it will end the current pass and proceed to the next pass. It will do this **3** times and displa
performance information for each pass. The *combined* performance of both devices is calculated and
displayed at the end of the run.

9.3 Example 3 – Time Stamping and Setup File

```
XDD –op write –targets 2 /dev/rdsk/dks10d2s0 /dev/rdsk/dks10d2s0
-setup XDD.setup –ts detailed –ts output ts.write
```

This test that will **read** sequentially from **2** targets that are actually a single device:
`/dev/rdsk/dks10d2s0.` The request size of **128** blocks at **2048** bytes per block, read limit of
4096 MegaBytes (4096 * 1024*1024 bytes), the time limit of **10** seconds for each pass, verbose outp
and pass count of **3** are all specified in the **XDD.setup** file which looks like so:

```
-blocksize 2048
–reqsize 128
-mbytes 4096
–verbose
–passes 3 -timelimit 10
```

The time stamp option is also used in this example to dump an ASCII output file called **ts.write.** It
should be noted that these time stamp file names are appended with a **t#** where **#** is the number of the
target that belongs to the particular time stamp file. In this example, since there are two targets, the ti
stamp files will be **ts.write.t0** and **ts.write.t1.**

9.4 Example 4 – Random Seeks

```
XDD –op read –targets 1 /dev/rdsk/dsk10d2s0 –reqsize 8
-mbytes 16 –passes 3 –verbose
–seek random –seek range 4000000
```

This is a very basic *random I/O* test that will **read** from target device **/dev/rdsk/dks10d2s0** starting a
some random location using a fixed request size of **8** blocks until it has read **16** MegaBytes (16 *

1024*1024 bytes). It will do this **3** times and display performance information for each pass. The def block size is 1024 bytes per block so the request size in bytes is 8 Kbytes (8 * 1024 bytes). The numb of requests that need to be generated to read 16 MegaBytes in 8192 byte chunks is 2048. Since this is purely random I/O pattern, these 2048 requests are distributed over a range of 4,000,000 blocks (agai 1024 bytes per block). This is useful in constraining the area over which the random locations are chosen from. The same seek locations are used for each pass in order to generate reproducible results fact, upon each invocation of XDD using the same parameters, the same random locations are genera each time. This allows the user to change the disk or starting offset or some such thing and observe th effects. The random locations may be changed from pass to pass within an XDD run by using the "-**randomize**" option in which case a new set of locations is generated for each pass. Furthermore, the random locations may be changed from run to run using the **–seek seed** option to specify a different random number generation seed value for each invocation of XDD.

9.5 Example 5 – End to End operation

Perform an end to end operation between two hosts, hostA and hostB, where hostA is the *source* and hostB is the *destination*.

Start the instance of XDD on hostB, the *destination* side first. This is required because if the *destinat* side is not running when the *source* side starts, the *source* side will terminate early because it will no able to connect to the instance of XDD on the *destination* side.

Hence, on the *destination* side:

```
      XDD –op write –targets 1 /tmp/foo2 –reqsize 4096 -mbytes 3000 –verbose \
      –e2e isdestination –e2e destination 192.168.17.10 –e2e port 2010
```

Once this is running it will wait for a connection from the *source* side before writing data to the *destination* target file **`/tmp/foo2.`**

On the *source* side:

```
      XDD –op read –targets 1 /tmp/foo1 –reqsize 4096 -mbytes 3000 –verbose \
      –e2e issource –e2e destination 192.168.17.10 –e2e port 2010
```

Once the *source* side starts, it will open a socket to the *destination* host and start reading the *source* f **`/tmp/foo1`** and sending it over the socket to the *destination* side.

Once the target file on the *source* side has been read and transferred, assuming no additional passes a requested, then the *source* side will terminate followed by the *destination* side. Each will display the usual results with an additional value at the end of each output line. This value will be a number betw 0 and 100 and represents the percentage of total time that was spent by XDD transferring data over th network.

It is important to follow these basic rules when running an e2e operation:

- The source and destination XDD command lines must contain either `–e2e issource` or `e2e isdestination` respectively in order to properly identify their respective roles
- The "operation" or –op for the *source* must be "read"
- The "operation" or –op for the *destination* must be "write"
- The number of megabytes to read from the *source* target should be equal to the number of megabytes written to the *destination* target
- The queue depth (i.e. –queuedepth option) must be the same on both the *source* and *destinati* sides for a given target
- The *destination* host name/address must be specified on both *source* and *destination XDD* command lines and must be the same

9.6 Example Time Stamp Output

```
Target and Qthread number for this report, 0, 0
IOIOIOIOIOIOIOIOIOIOI XDD version Linux.7.0.0.rc9.012810.Build.0317 based on Linux.7.0.0.rc9.011810.Build.1412 IOIOIOIOIOIOIOIOIOIOI
xdd - I/O Performance Inc., US DoE/DoD Extreme Scale Systems Center <ESSC> at Oak Ridge National Labs <ORNL> - Copyright 1992-2010

XDD DISCLAIMER:
 *** >>>> WARNING <<<<
 *** THIS PROGRAM CAN DESTROY DATA
 *** USE AT YOUR OWN RISK
 *** IOPERFORMANCE and/or THE AUTHORS ARE NOT LIABLE FOR
 *** >>>> ANYTHING BAD <<<<
 **** THAT HAPPENS WHEN YOU RUN THIS PROGRAM
     ...although we will take credit for anything good that happens
     but we are not *liable* for that either.


Starting time for this run, Thu Jan 28 03:22:28 2010

ID for this run, 'No ID Specified'
Maximum Process Priority, disabled
Passes, 1
Pass Delay in seconds, 0
Maximum Error Threshold, 16
Target Offset, 0
I/O Synchronization, 0
Total run-time limit in seconds, 0
Output file name, test-run.txt
CSV output file name, test-run.csv
Error output file name, stderr
Pass synchronization barriers, enabled
Number of Targets, 1
Number of I/O Threads, 1

Computer Name, pod7.ccs.ornl.gov, User Name, tmruwart
OS release and version, Linux 2.6.18-128.1.6.el5 #1 SMP Wed Apr 1 06:58:14 EDT 2009
Machine hardware type, x86_64
Number of processors on this system, 8
Page size in bytes, 4096
Number of physical pages, 8241895
Megabytes of physical memory, 32194
Clock Ticks per second, 100
Seconds before starting, 0
       Target[0] Q[0], /data/xfs/testfile
               Target directory, "./"
               Process ID, 2523
               Thread ID, 2524
               Processor, all/any
               Read/write ratio,  0.00 READ, 100.00 WRITE
               Throttle in MB/sec,   0.00
               Per-pass time limit in seconds, 0
               Pass seek randomization, disabled
               File write synchronization, disabled
```

```
                Request size, 4096, blocks, 4194304, bytes
                Number of Operations, 16, of, 16, target ops for all qthreads
                Start offset, 0, blocks, 0, bytes, 0.000 KBytes, 0.000 MBytes, 0.000 GBytes, 0.000000 TBytes
                Total data transfer for this target, 65536, blocks, 67108864, bytes, 65536.000 KBytes, 64.000 MBytes, 0.062 GBytes, 0.000061 TBytes
                Total data transfer for this QTHREAD, 65536,blocks, 67108864, bytes, 65536.000 KBytes, 64.000 MBytes, 0.062 GBytes, 0.000061 TBytes
                Pass Offset, 0, blocks, 0, bytes,                                        0.000 KBytes,  0.000 MBytes, 0.000 GBytes, 0.000000 TBytes
                Seek range, 1048576, blocks,                  1073741824, bytes, 1048576.000 KBytes, 1024.000 MBytes, 1.000 GBytes, 0.000977 TBytes
                Seek pattern, sequential
                Flushwrite interval, 0
                I/O memory buffer is a normal memory buffer
                I/O memory buffer alignment in bytes, 4096
                Data pattern in buffer,0x00
                Data buffer verification is disabled.
                Direct I/O, enabled
                Preallocation, 0
                Queue Depth, 1
                Timestamping, enabled for DETAILED SUMMARY
                Timestamp ASCII output file name, test-run.target.0000.qthread.0000.csv
                Delete file, disabled

Start of DETAILED Time Stamp Report, Number of entries, 16, Picoseconds per clock tick, 1000000, delta, 0
Target, RWV Op, Pass, OP Number, Block Location, Distance, StartTS, EndTS, IO_Time_ms, Relative_ms, Delta_us, Loop_ms, Inst MB/sec
0,w,1,0,0,0,10288350303715490112,10288350437959490112,         134.24400,        0.00000,        0.00000,        0.00000,      31.244
0,w,1,1,4096,0,10288350437966490112,10288350478994490112,       41.02800,      134.25100,        7.00000,       41.03500,     102.230
0,w,1,2,8192,0,10288350478997490112,10288350533440490112,       54.44300,      175.28200,        3.00000,       54.44600,      77.040
0,w,1,3,12288,0,10288350533446490112,10288350736743490112,     203.29700,      229.73100,        6.00000,      203.30300,      20.631
0,w,1,4,16384,0,10288350736750490112,10288350789934490112,      53.18400,      433.03500,        7.00000,       53.19100,      78.864
0,w,1,5,20480,0,10288350789937490112,10288350839706490112,      49.76900,      486.22200,        3.00000,       49.77200,      84.275
0,w,1,6,24576,0,10288350839708490112,10288350973728490112,     134.02000,      535.99300,        2.00000,      134.02200,      31.296
0,w,1,7,28672,0,10288350973733490112,10288351026693490112,      52.96000,      670.01800,        5.00000,       52.96500,      79.198
0,w,1,8,32768,0,10288351026697490112,10288351109349490112,      82.65200,      722.98200,        4.00000,       82.65600,      50.747
0,w,1,9,36864,0,10288351109354490112,10288351173507490112,      64.15300,      805.63900,        5.00000,       64.15800,      65.380
0,w,1,10,40960,0,10288351173509490112,10288351250697490112,     77.18800,      869.79400,        2.00000,       77.19000,      54.339
0,w,1,11,45056,0,10288351250703490112,10288351290625490112,     39.92200,      946.98800,        6.00000,       39.92800,     105.062
0,w,1,12,49152,0,10288351290627490112,10288351353374490112,     62.74700,      986.91200,        2.00000,       62.74900,      66.845
0,w,1,13,53248,0,10288351353380490112,10288351431280490112,     77.90000,     1049.66500,        6.00000,       77.90600,      53.842
0,w,1,14,57344,0,10288351431284490112,10288351621563490112,    190.27900,     1127.56900,        4.00000,      190.28300,      22.043
0,w,1,15,61440,0,10288351621569490112,10288351678337490112,     56.76800,     1317.85400,        6.00000,       56.77400,      73.885
End of DETAILED Time Stamp Report
Start of SUMMARY Time Stamp Report
Average seek distance in 1024 byte blocks, 0, request size in blocks, 4096
Range:  Longest seek distance in blocks, 0, shortest seek distance in blocks, 0
Average I/O time in milliseconds,        77.51937, average dead time in microseconds,         4.53333
I/O Time Range:  Longest I/O time in milliseconds,       203.29700, shortest I/O time in milliseconds,       39.92200
Dead Time Range:  Longest dead time in microseconds,         7.00000, shortest dead time in microseconds,         2.00000
End of SUMMARY Time Stamp Report
```

Under the Hood

This section is a look at XDD program organization and data structures. This section is primarily here for the author’s benefit because his brain is getting old and forgetful.

Starting with XDD6.3 the source code files have been separated out into several categories of source files. Th process of moving functions around within the files is mostly complete but further changes may be made in fu releases. The rationale behind this change is that some of the source files were getting too large. The categorie currently defined are

Initialization:
- initialization.c
- parse.c
- parse_func.c
- parse_table.c
- global_data.c
- ptds.c
- schedule.c
- processor.c
- io_buffers.c
- datapatterns.c
- preallocate.c
- signals.c

Reporting:
- results_display.c
- results_manager.c
- info_display.c
- timestamp.c
- heartbeat.c
- restart.c

Timing and clock functions:
- global_time.c
- global_clock.c
- pclk.c
- ticker.c

Core functions:
- access_pattern.c
- barrier.c
- end_to_end.c
- qthread.c
- qthread_init.c
- qthread_cleanup.c
- qthread_io.c
- qthread_ttd_*.c
- read_after_write.c
- target_thread.c
- target_init.c
- target_cleanup.c
- target_pass.c
- target_ttd_*.c
- lockstep.c
- utils.c
- xdd.c

Special functions: nt_compat.c
Other programs: timeserver: timeserver.c and gettime: gettime.c

9.7 XDD general operation

XDD is a command-line driven program. The first thing it does upon invoking it is to parse the command line
the command line contains the "-setup" option, it will parse the command line options from left to right up to
–setup option. It will then parse the setup file options followed by parsing the remaining command line option
After the all the requested options have been set, the main XDD program will start all the appropriate threads
at a time. Each thread will go through it initialization phase which includes the following:

- Open the target and prepare it for access
- Generate the list of locations to access and the associated access pattern and operations
- Allocate I/O buffer
- Perform timestamp setup if necessary
- Display target-specific information as requested

After the thread has completed its setup process, it will enter a serialization barrier that will release the main X
parent thread which will in turn start the next thread. The target thread that has just completed initialization wi
enter the main barrier waiting for all the other target threads to complete initialization. Once all the threads ha
been started, the last thread to start will enter the starting barrier and cause all the threads to be released and th
fun begins.

All the threads will do their respective I/O operations for a single pass and then enter the pass barrier. The pas
barrier causes threads to wait until all threads have completed a pass before starting the next pass. If this is the
pass, then upon being released from this barrier, all the threads will perform any clean-up operations and exit.

Another function of the pass barrier is to hold all the threads dormant whilst thread 0 gathers all the results
information from each of the threads and generates the appropriate intermediate results and displays them if

requested. This is also the case when all threads have completed all passes and thread 0 needs to process sumr
information as well.

9.8 The XDD buffer memory layout

XDD uses several buffers during a normal run. All buffers are page aligned. These buffers include:

- The Read/Write I/O buffer. This buffer is simply the size of a single I/O request and there is one of th buffers per target being tested. This buffer is normally allocated from user space memory but if the "-sharedmemory" option is specified the buffer will be allocated from a shared memory segment. The u of a shared memory segment can have a significant impact on bandwidth performance in some cases.
- Access location / operation buffer (aka seeklist). This buffer is variable in size and depends on the nur of I/O operations that need to be performed. There is one access location/operation buffer for each tar being tested. This buffer contains the following information:
  - The block location to access
  - The operation to be performed (read or write)
  - The amount of data to transfer for that operation
  - The time to issue the I/O request
- Time stamp buffer if time stamping is enabled. This buffer can get very large because there is one tim stamp entry for every I/O operation to be performed for the entire XDD run (which includes all passes For example, if there are three targets and each target will perform 8192 operations per pass and there 4 passes, then there will be three time stamp buffers (one for each target) with 32768 entries in each buffer (4 passes times 8192 entries per pass). Each time stamp entry contains the following informatic
  - Start time stamp (64-bit)
  - End time stamp (64-bit)
  - Operation (read or write)
  - Amount of data transferred
  - Pass number
  - Operation number
- Per Thread Data Structure (ptds). This data structure contains all the information related to a single tar during an XDD run. This is explained in more detail in the section on the *XDD thread structure*.

9.9 The XDD thread structure

XDD uses POSIX threads for all processes that it controls. There are one or more threads per target. Each targ will have a primary thread and potentially some number of Q threads. The Q threads are used to perform paral asynchronous I/O on a particular target. The concept of Q threads is essentially "aio" but since "aio" is differe from system to system, it was easier to simply implement one within XDD.

Each thread (primary or Q thread) has a PTDS (per thread data structure) data structure associated with it. The PTDS contains all the information related to a thread running at any given time. This structure is passed to the various routines within XDD. The PTDS's are allocated on an as-needed basis. The layout of the PTDS struct is shown in Figure 1.

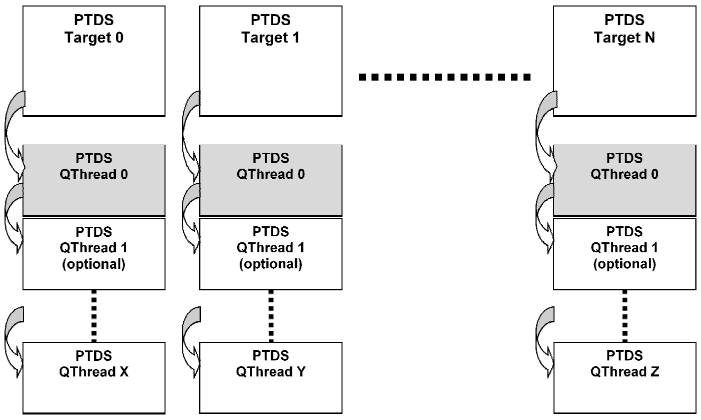


Figure 2 The Per Target Data Structure (PTDS) layout including the QThreads.

In the above diagram there are N targets each with some number of QThreads. The number of QThreads is eq to the Queue Depth specified for each target (using the –queuedepth option). The number of QThreads can be different for each target as is shown in this diagram but in practice the number of QThreads is the same for all targets during a run. The primary PTDS for each target is the one owned by the Target Thread. There is a minimum of one QThread for each Target Thread and the subsequent QThreads are chained together as shown the diagram. The last QThread in the chain is identified by the fact that its “next QThread” point is null.

9.10 XDD barriers

XDD uses a number of synchronization barriers in order to provide precise control and certain functionality su as lock-stepping. The primary barriers cause all the threads to begin execution at precisely the same time for e pass of a run. As of XDD version 7.0 the barriers are implemented using pthread barriers, ptherad mutexes, an pthread semaphores. (Older versions of XDD used System V semaphores which tended to leave semaphores allocated after being canceled or killed.) The amount of time required to process a semaphore request is significantly less than the time of an I/O operation and has no measurable effect on the results.

10 The GNU Public License

10.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--
sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free S
Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this servi
wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copi
software, or if you modify it.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code.
must show them these terms so they know their rights.
We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.
Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its re
to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.
Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprie
prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.
The precise terms and conditions for copying, distribution and modification follow.

10.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refe
such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim
modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered o
contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.
1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright no
disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.
You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provi
you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties u
terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an annou
including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and te
user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to
announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves,
License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distributio
whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works base
Program.
In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work u
scope of this License.
3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for
interchange; or,
- **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete r
readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and onl
received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any as
interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distrib
either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the sour
even though third parties are not compelled to copy the source along with the object code.
4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will autom
terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by la
do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for
distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these te
conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherw
contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other

10.3 NO WARRANTY

11 Acknowledgements

HPUX is a registered trademark of Hewlett-Packard, Inc.
AIX is a registered trademark of IBM Corporation.
IRIX is a registered trademark of SGI.
Solaris and SPARC are registered trademarks of Sun Microsystems, Inc.
Windows, Windows NT, and Windows 2000 are registered trademarks of Microsoft, Inc.

The author would like to acknowledge and thank all those people who have helped with adding various functi
and bug fixes to XDD over the years.

The recent work on XDD version 6.7 and beyond is supported by the Oak Ridge National Laboratory Extreme Scale System Center and the United States Departme of Defense.

XDD Command Line Arguments Cheat Sheet

-blocksize *[target <target#>] number_of_bytes_per_block*
-bytes *[target <target#>] number_of_bytes_to_xfer_per_pass*
-combined *filename*
-createnewfiles *[target #]*
-csvout *filename*
-datapattern *[target <target#>] character_pattern –or-*
*"***random***" –or-*
*"***sequenced***" –or-*
*"***prefix***" –or-*
*"***inverse***" –or-*
"**ascii** <*string*>" *–or-*
"**hex** <*hex digits 0-9, a-f or A-F*>*" –or-*
"**replicate***" –or-*
"**lfpat***" –or-*
"**ltpat***" –or-*
"**cjtpat***" –or-*
"**crpat***" –or-*
"**cspat***"*
-deletefile *[target <target#>]*
-deskew
-dio *[target <target#>]*
-endtoend *[target <target#>]*
issource | isdestination
port *#*
destination *hostname-or-IPaddress*
-errout *filename*
-flushwrite *#ops*

-fullhelp
-heartbeat *#seconds*
-help *option_name*
-id **commandline** - or - *"id_string"*
-kbytes *[target <target#>] number_of_kilobytes_to_transfer*
-lockstep <*master_target*> <*slave_target*>
<*when*> <*howlong*>
<*what*> <*howmuch*>
<*startup*> <*completion*>
-maxall
-maxerrors *number_of_errors*
-maxerrorstoprint *number_of_errors_to_print*
-maxpri
-mbytes *[target <target#>] number_of_megabytes_to_transfer*
-memalign *[target <target#>] alignment_value_in_bytes*
-minall
-nobarrier
-nomemlock
-noproclock
-numreqs *[target <target#>] number_of_requests_to_perform*
-op *[target <target#>] read*|*write*
-output *filename*
-outputformat **add** | **new** *<format_id_string>*
-passdelay *seconds*
-passes *number_of_passes*
-passoffset *[target <target#>] offset_in_blocks*
-percentcpu **absolute | relative**
-preallocate *[target <target#>] number_of_bytes*
-processlock
-processor *processor_number target_number*
-rwratio *[target <target#>] %read*
-queuedepth *[target <target#>] number_of_commands_per_target*
-qthreadinfo
-randomize *[target <target#>]*
-recreatefiles *[target #]*
-reopen *[target #]*
-reportthreshold *[target #] <#.#>*
-reqsize *[target <target#>] number_of_blocks*
-restart *[target <target#>]*
enable
frequency *<seconds>*
file *<name_of_restart_file>*
offset *<offset_in_bytes>*
-roundrobin #
-runtime *seconds*
-rwratio *[target #] <readwriteratio>*
-seek *[target <target#>]*
save *filename*
load *filename*
disthist *#*
seekhist *#*
random
range #
stagger
interleave #
seed #
none
-setup *setup_filename*
-sgio
-sharedmemory *[target <target#>]*
-singleproc *processor_number*
-startdelay *#seconds*
-startoffset *[target <target#>] starting_block_number*
-starttime *#seconds*
-starttrigger *targetA target>*
time|op|percent|mbytes|kbytes *#*
-stoponerror
-stoptrigger *targetA target>*
time|op|percent|mbytes|kbytes
-syncio *number*
-syncwrite *[target <target#>] number*
-target *filename*
-targetdir *[target <target#>] directoryname pass*
-targetoffset *[target <target#> offset_in_blocks*
-targets *N filename1 filename2 filenameN*
-targetstartdelay *#seconds_multiplier*
-throttle *[target <target#>]*
ops *operations/sec*
bw *megabytes/second*
-timelimit *[target <target#>] seconds_per_pass*
-timeserver
host *hostname*
port *port#*
bounce *bounce_count*
-timestamps *[target <target#>*
output *output_filename_pre*
summary
detailed
normalize
summary
wrap
oneshot
size #
triggertime *#seconds*
triggerop *operation#*
append
dump *dump_filename_prefi*
-verbose
-verify *[target <target#>]*
location
contents
-version

Output Format Identifiers Cheat Sheet

```
+WHAT
+PASS
+TARGET
+QUEUE
+BYTESTRANSFERED
+BYTESXFERED
+BYTESREAD
+BYTESWRITTEN
+OPS
+READOPS
+WRITEOPS
+BANDWIDTH
+READBANDWIDTH
+WRITEBANDWIDTH
+IOPS
+READIOPS
+WRITEIOPS
+LATENCY
+ELAPSEDTIME1STOP
+ELAPSEDTIMEPASS
+OVERHEADTIME
+PATTERNFILLTIME
+BUFFERFLUSHTIME
+CPUTIME
+PERCENTCPUTIME
+PERCENTCPU
+USERTIME
+USER
+USRTIME
+SYSTEMTIME
+SYSTEM
+SYSTIME
+PERCENTUSER
+PERCENTUSERTIME
+PERCENTUSR
+PERCENTSYSTEM
+PERCENTSYSTEMTIME
+PERCENTSYS
+OPTYPE
+XFERSIZEBYTES
+XFERSIZEBLOCKS
+XFERSIZEKBYTES
+XFERSIZEMBYTES
+E2EIOTIME
+E2ESRTIME
+E2EPERCENTSRTIME
+E2ELAGTIME
+E2EPERCENTLAGTIME
+E2EFIRSTREADTIME
+E2ELASTWRITETIME
+DELIM
```